VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

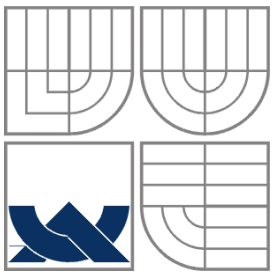
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE OBLIČEJŮ VE VIDEU

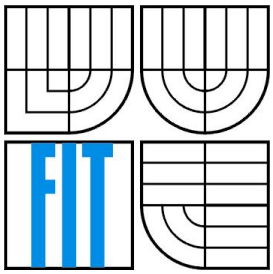
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. ALEŠ KOLMAN
AUTHOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE OBLIČEJŮ VE VIDEU

FACE DETECTION IN VIDEO

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ALEŠ KOLMAN

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LUKÁŠ POLOK

BRNO 2012

Zadání diplomové práce/12134/2011/xkolma02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2011/2012

Zadání diplomové práce

Řešitel: **Kolman Aleš, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Detekce obličejů ve videu**
Face Detection in Video
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte a popište algoritmy zpracování obrazu a detekce objektů v obraze, zejména pak obličejů.
2. Prostudujte metody optimalizací daných algoritmů.
3. Navrhněte knihovnu pro detekci obličejů ve videu, pracující v reálném čase.
4. Implementujte navrženou knihovnu s důrazem na použitelnost, implementujte jednoduchou demonstrační aplikaci.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti budoucího vývoje.
6. Vytvořte stručný plakát prezentující vaši práci, její cíle a výsledky.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1-3.

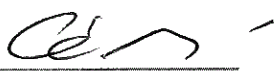
Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Polok Lukáš, Ing.**, UPGM FIT VUT
Datum zadání: 19. září 2011
Datum odevzdání: 23. května 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2



doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Práce se zabývá problematikou detekce obličejů ve videu. Naleznete v ní souhrn základních barevných modelů. Poté je uveden popis a srovnání základních metod pro detekci lidské kůže s praktickou ukázkou implementace parametrického detektoru. Následuje teoretický základ pro detekci obličejů a sledování obličejů ve videu obsahující výčet základních pojmů a metod této problematiky. Vyšší důraz je kladen na popis algoritmu strojového učení AdaBoost a na popis možnosti použití Kalmanova filtru pro účely sledování detekovaných obličejů. V podlesní části práce jsou uvedeny návrh, implementace a testování knihovny realizované v rámci této diplomové práce.

Klíčová slova

Detekce obličejů, sledování obličejů, obraz, barevný model, detekce kůže, klasifikátor, kaskáda, Haarovy příznaky, integrální obraz, AdaBoost, Kalmanův filtr, LBP, SVM, neuronové sítě, OpenCV

Abstract

The project is focused on face detection in video. Firstly, it contains a summary of basic color models. Secondly, you can find the description and comparison of the basic methods for detection of human skin with a practical example of implementation of parametric detector. Thirdly, a theoretical basis for face detection and face tracking in a video containing a list of basic concepts and methods of this issue follows. Greater emphasis is placed on the description of machine learning algorithm AdaBoost and description of the possible application of the Kalman filter for the purpose of face tracking. Design, implementation and testing of library accomplished within the master thesis are listed in the final part of this thesis.

Keywords

Face detection, face tracking, image, color model, skin detection, classifier, cascade, Haar features, integral image, AdaBoost, Kalman filter, LBP, SVM, neural networks, OpenCV

Citace

Kolman Aleš: Detekce obličejů ve videu. Brno, 2012, diplomová práce, FIT VUT v Brně.

Detekce obličejů ve videu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Lukáše Poloka
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Aleš Kolman
22. 5. 2012

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce Ing. Lukáši Polokovi za odborné vedení a udílení cenných rad. Také bych rád poděkoval Soně Kolmanové a Eleně Novákové za trpělivost a důslednost při korekturách a opravách.

Obsah

1 Úvod

Hlavní náplní této práce je popis problematiky detekce obličejů ve videu. Mezi vedlejší témata popsaná v rámci této práce pak spadá detekce lidské kůže ve videu a sledování detekovaných tváři ve videu. Všechna zmíněná témata spadají do kategorie zpracování obrazu. Samotná kategorie zpracování obrazu pak v hierarchii dále spadá pod počítačovou grafiku.

Druhá kapitola obsahuje definice základních pojmů počítačové grafiky, jako je např. obraz, či barevný model. Dále jsou ve druhé kapitole blíže popsány vybrané barevné modely, mimo jiné nechybí popis velmi důležitého barevného modelu RGB, ani popis barevného modelu CMYK, jenž je používán především v tiskárnách. Pak je uveden popis modelu HLS, který nejlépe odráží lidský přístup k míchání barev. V této kapitole je ještě uveden barevný model YC_RC_B, jenž se vyznačuje oddělením jasovou složky od barevných. Ke konci druhé kapitoly je rozebrán princip reprezentace obrazu ve stupních šedi se vzorci pro převody barevných modelů do této reprezentace obrazu.

Ve třetí kapitole je uveden kompletní rozbor problematiky detekce lidské kůže v obraze. Nejprve jsou popsány základní metody detekce lidské kůže. Poté jsou tyto metody rozděleny do tří skupin na explicitně definované metody, neparametrické metody a parametrické metody. Dále je uvedena praktická implementace detektoru lidské kůže v obraze. Konkrétně je uvedena praktická implementace pouze parametrického detektoru lidské kůže v obraze využívajícího jednu Gaussovu křivku pro matematický popis. Implementace explicitního detektoru není třeba uvádět, jelikož implementace explicitního detektoru je nesmírně jednoduchá a přímo plyne z teoretického základu explicitních metod. Implementace neparametrického detektoru pak není uvedena z důvodu velmi sporadického nasazování neparametrických metod do praxe a z důvodu omezeného rozsahu této práce. Dále ve třetí kapitole je uvedeno srovnání všech detektorů lidské kůže vyzkoušených v rámci diplomové práce. Nakonec třetí kapitoly je provedeno zhodnocení všech popisovaných metod pro detekci kůže s ohledem na možnost jejich nasazení pro detekci obličejů ve videu.

Čtvrtá kapitola podrobně popisuje teoretický základ pro detekci obličejů ve videu a okrajově popisuje i teoretický základ pro sledování obličejů ve videu. Ve čtvrté kapitole jsou nejprve vysvětleny stěžejní pojmy používané s detekcí a sledováním obličejů ve videu. Dále je ve čtvrté kapitole uveden výčet významných metod spojených s detekcí obličejů ve videu, přičemž je vhodné poznamenat, že nebylo možné postihnout všechny zajímavé metody spojené s detekcí lidských tváří v obraze. Do čtvrté kapitoly tak byly zahrnuty jen metody jistým způsobem průlomové nebo jedinečné a metody, jejichž principy přímo ovlivnily návrh výsledného detektoru lidských tváří. Tato kritéria splnily metoda strojového učení AdaBoost, metoda využívající neuronových sítí, metoda SVM a metoda využívající LBP. Na konec čtvrté kapitoly jsou pak uvedeny metody spojené se sledováním objektů v obraze, resp. pouze jedna metoda spojená se sledováním obličejů v obraze a to metoda založená na Kalmanově filtru.

V páté kapitole je uveden návrh řešení výsledné knihovny. Pátá kapitola začíná představením návrhu funkce pro detekci obličejů ve videu, zmíněná funkce je součástí výsledné knihovny. Tato funkce provádí detekci tváří pomocí klasifikátoru natrénovaného přímo odborníky stojícími za grafickou knihovnou OpenCV. Podrobný popis struktury zmíněného OpenCV klasifikátoru je uveden hned v další části páté kapitoly. Poté následuje popis principu standardní klasifikace OpenCV klasifikátorem, ze kterého vyplyne, že OpenCV klasifikátor je velice robustní a velice pomalý. Dále pak následují návrhy optimalizací, jež představují možnosti kompenzace nízké rychlosti OpenCV klasifikátoru. Mezi zmíněné optimalizace mimo jiné patří i optimalizace založené na použití SIMD instrukcí, konkrétně SSE II instrukcích. V páté kapitole je poté představen návrh další funkce z výsledné knihovny, konkrétně funkce realizující sledování tváří ve videu s využitím Kalmanova filtru. Na konec páté kapitoly jsou uvedeny dva návrhy nejvhodnějšího spojení obou zmíněných funkcí pro účely souběžného sledování a detekování tváří ve videu, kdy oba tyto návrhy vychází z vícevláknového běhu.

Šestá kapitola popisuje praktickou implementaci výsledné knihovny a ukázkových aplikací. Šestá kapitola začíná popisem programového rozhraní knihovny pro detekci obličejů ve videu. Poté následuje popis programového rozhraní knihovny pro sledování tváří v obraze. Závěrem šesté kapitoly jsou pak uvedeny implementace dvou ukázkových aplikací, jež presentují vhodný způsob použití výsledné knihovny pro souběžné sledování a detekování tváří ve videu.

V sedmé kapitole jsou popsány průběhy testování všech navržených, resp. implementovaných řešení. Nejprve jsou ve zmíněné kapitole uvedeny výsledky testů OpenCV klasifikátorů. Dále jsou uvedeny výsledky testování detekce tváří v obraze s různým stupněm optimalizace. Poté jsou uvedeny výsledky testů souběžného sledování a detekování tváří ve videu. Na konec sedmé kapitoly je pak provedeno zhodnocení výsledků ze všech testů.

V závěru celé práce jsou uvedeny poznatky získané v rámci řešení celé diplomové práce, celkový přínos diplomové práce, zhodnocení výsledné knihovny s ohledem na zadání diplomové práce a návrhy na možná rozšíření či zlepšení výsledné knihovny.

Obrázek 1.1: Detekce obličejů fotoaparátem od společnosti SONY [29].

2 Obraz a jeho reprezentace

Pokud je zmíněn pojem obraz, bez bližší specifikace tohoto pojmu, těžko si všichni představí stejnou věc, proto je vhodné si napřed zavést definici pro tento pojem. Definovat obraz není ale tak snadné, jelikož existuje celá řada definic, které se od sebe liší a které závisí na konkrétní disciplíně, pro kterou je definice prováděna. V tomto textu bude obraz chápán ve smyslu definice publikované v [1], jež obraz definuje jako spojitou funkci dvou proměnných:

$$z = f(x, y), \tag{2.1}$$

této funkci se obecně říká obrazová funkce. V reálných reprezentacích má obraz vždy omezené rozměry, toho lze využít k vymezení definičního oboru obrazové funkce pomocí kartézského součinu dvou spojitých intervalů z oboru reálných čísel, reflektující rozměry obrazu. Aplikování předešlého poznatku můžeme potom psát, že funkce realizuje zobrazení [1]:

$$f : \left(\langle x_{\min}, x_{\max}\rangle \times \langle y_{\min}, y_{\max}\rangle\right) \to H, \tag{2.2}$$

kde *x*, *y* jsou souřadnice bodu ve dvou rozměrech, v nichž funkce nabývá hodnoty *z*, kde $z \in H$, přičemž platí že *x* patří do intervalu $\langle x_{\min}, x_{\max}\rangle$ a současně *y* náleží do intervalu $\langle y_{\min}, y_{\max}\rangle$. Tento zápis však říká, že hodnotou obrazové funkce musí být pouze jedno jediné reálné číslo, což v počítačové grafice není vždy dostačující, například pro reprezentaci obrazu v barevném modelu RGB, je potřeba ke každé kombinaci *x*, *y* přiřadit tří složkový vektor [R, G, B] (model RGB bude probrán později v této kapitole). Proto je pro obrazovou funkci používán také následující zápis [1]:

$$f : \left(\langle x_{\min}, x_{\max}\rangle \times \langle y_{\min}, y_{\max}\rangle\right) \to \left(H_1 \times H_2 \times ... \times H_n\right), \tag{2.3}$$

jenž říká, že obrazová funkce může nabývat hodnot zapsaných jako uspořádaná *n*-tice údajů $z = [z_1, z_2, ..., z_n]$, resp. $z \in \left(H_1 \times H_2 \times ... \times H_n\right)$.

V oblasti počítačové grafiky má obraz zřídkakdy spojitý definiční obor funkce, místo toho se daleko častěji používá nějaká forma diskrétního definičního oboru, resp. rastru. Rastr je tvořen obrazovými prvky, jež se v počítačové grafice nazývají *pixely* (zkratka z anglického **pi**cture **el**ement).

2.1 Barevné modely

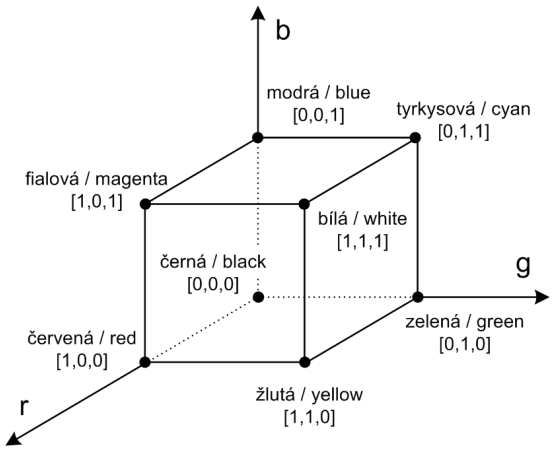
V dnešní době existuje několik barevných modelů pro reprezentaci obrazu, přičemž různé modely nachází uplatnění v různých činnostech, snímání obrazu, zobrazení obrazu, přenos obrazu, převod obrazu, atd. nemůžeme tedy říct, že by existoval jeden barevný model, který by nejlépe vyhovoval všem činnostem, které mohou nastat. Můžu zde pouze poznamenat, že nejpoužívanějším barevným modelem v dnešní době je model RGB [1]. Následuje výčet těch nejdůležitějších barevných modelů.

2.2 RGB, RGBA

Model RGB je požíván především pro zobrazovací zařízení, např. monitor. RGB prostor využívá tři základní barvy, červenou (***r**ed*), zelenou (***g**reen*) a modrou (***b**lue*). Mícháním těchto tři barev je teoreticky možné zobrazit jakoukoliv barvu vyskytující se v reálném světě. Prakticky nás však limituje počet bitů, které použijeme pro reprezentaci dané barvy. Použijeme-li k reprezentování barvy pouze jeden bit, budeme schopni reprezentovat pouze dvě barvy, použijeme-li jeden bajt, budeme schopni reprezentovat 256 barev, atd. RGB model k reprezentování barev obecně používá tří složkový vektor, jehož složky nabývají hodnot z intervalu <0,1> [1], kde 0 značí, že složka se ve výsledné barvě nevyskytuje a 1 naopak značí, že daná složka je ve výsledné barvě zastoupená ve své nejvyšší možné intenzitě. Při všech složkách vektoru nastavených na 0 dostaneme černou barvu, nastavením všech složek na 1 zobrazíme bílou barvu.

Počet bitů přidělených jednotlivým složkám vektoru nám určuje rozlišovací schopnost zobrazení. Mezi nejčastěji používané reprezentace patří reprezentace barev na jednom bitu, kdy 0 odpovídá černá barva a 1 bílá, reprezentace na jednom bytu, kdy je daný byte použit buď k reprezentaci 256 stupňům šedi, anebo k reprezentování r, g a b složek v poměru 3 bity pro červenou, 3 bity pro zelenou a 2 bity pro modrou. Pro modrou je vždy alokováno nejméně prostoru a naopak pro zelenou nejvíce, tento nepoměr vychází z fyziologických vlastností lidského oka. Dalšími významnými paměťovými schématy pro reprezentaci RGB modelu je reprezentace na 16 bitech, kdy je pro reprezentaci r/g/b složek použito 5/6/5 bitů, toto schéma nese název *high color*, reprezentace na 24 bitech v poměru 8/8/8, toto schéma nese název *true color* a konečně posledním důležitým schématem je RGBA reprezentované na 32 bitech, které přidává do vektoru čtvrtou složku *alpha* určující průhlednost výsledné barvy, 0 pro minimální průhlednost a 1 naopak pro maximálně průhlednou barvu, rozdělení bitů je 8/8/8/8 pro R/G/B/α. Schéma *true color* je dnes nejpoužívanější [1] a umožňuje zobrazit 256^3 = 16 777 216 barev.

Barevný rozsah RGB je často zobrazován do 3D prostoru pomocí jednotkové krychle. Tomuto zobrazení se říká geometrická reprezentace a je uvedeno na následujícím obrázku 2.1.



Obrázek 2.1: Geometrická reprezentace prostoru RGB [1].

Jednotková krychle z obrázku 2.1 je konkrétně vyobrazena tak, že osám *x*, *y*, *z* odpovídají složky r, g, b. Počátek souřadnic, vektor [0, 0, 0] odpovídá černé barvě a vektor [1, 1, 1] odpovídá bílé barvě. Vrcholy krychle ležící na osách r, g, b odpovídají barvám r, g, b o maximální intenzitě a konečně zbylé vrcholy odpovídají barvám c, m, y, o nichž bude řeč dále v textu.

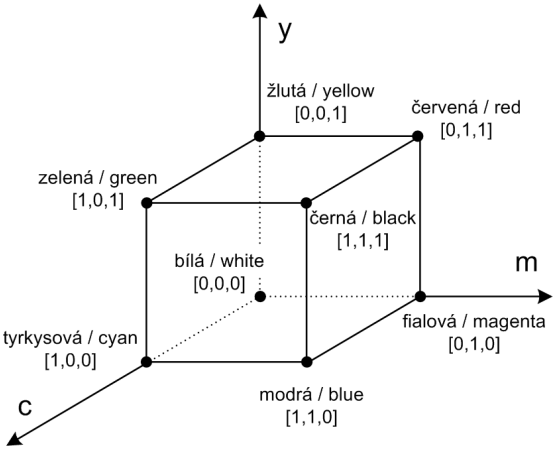
2.3 CMY, CMYK

Model CMY je požíván především pro reprodukční zařízení, např. tiskárna [2]. CMY prostor využívá tří základní barvy, tyrkysovou (***c****yan*), fialovou (***m****agenta*) a žlutou (***y****ellow*). Stejně jako v RGB záleží počtu bitů použitých pro reprezentování barvy, čím více bitů, tím více odstínů barev jsme schopni dostat. Stejně jako RGB model i CMY k reprezentování barev obecně používá tří složkový vektor, jehož složky nabývají hodnot z intervalu <0,1> [1], kde 0 značí, že složka se ve výsledné barvě nevyskytuje a 1 naopak značí, že daná složka je ve výsledné barvě zastoupená ve své nejvyšší možné intenzitě. Při všech složkách vektoru nastavených na 0 dostaneme bílou barvu, nastavením všech složek na 1 vytiskneme černou barvu. Tiskárny však černou barvu nezískávají mícháním, protože míchání černé barvy ze tří jiných není ekonomické a protože umícháním všech tří barev dohromady by vznikla pouze velmi tmavě hnědá, nikoliv však čistě černá barva. Základní tři barvy v tiskárně totiž z technologického principu tisku nemohou dosahovat plného krytí jako v teoretickém modelu.

Proto vznikl model CMY obohacený o černou (*blac****k***) složku, který nese zkratku CMYK. Velikost černé složky pro danou barvu získáme jako minimální hodnotu ze složek c, m a y, které poté snížíme o k [1]. V praxi je tento vzorec přizpůsobován krycím vlastnostem tiskových barev.
Převod mezi prostory RGB a CMY je řízen dle následujícího vztahu [1]:

$$\begin{bmatrix} c \\ m \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} r \\ g \\ b \end{bmatrix}. \qquad (2.4)$$

Barevný rozsah CMY je stejně jako model RGB často zobrazován do 3D prostoru pomocí jednotkové krychle, obrázek 2.2, kdy osám *x*, *y*, *z* odpovídají složky c, m a y.

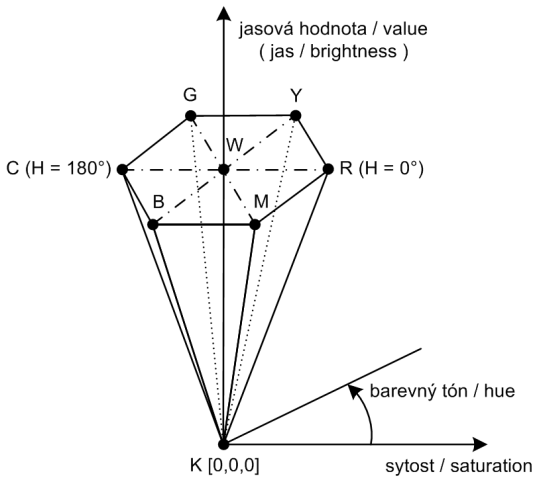


Obrázek 2.2: Geometrická reprezentace prostoru CMY [1].

2.4 HSV, HSB

Model HSV, někdy také nazývaný HSB, nejvíce reflektuje lidský přístup ke vnímání barev. Tyto modely jsou tvořeny třemi hlavními parametry, barevný tón (***h**ue*), sytost (***s**aturation*) a jasová hodnota (***v**alue*) v případě HSV, resp. jas (***b**rightness*) v případě HSB. Barevný tón je reprezentován ve stupních od 0° do 360° a značí základní spektrální barvu, sytost určuje míru základního odstínu vůči šedé a nabývá hodnot <0,1>, kdy 0 znamená žádné zastoupení základního odstínu, tzn., že výsledná barva bude odstínem šedé barvy, 1 naopak znamená, že výsledná barva bude odstínem pouze čistého základního tónu barvy, bez příměsí šedé. Jasová hodnota určuje světlost výsledné barvy, resp. podíl bílého (bezbarvého) světla ve výsledku, nabývá hodnot <0,1>, kdy 0 znamená maximálně tmavou barvu a 1 naopak maximálně světlou barvu. Z předchozího vyplývá, že při nastavení sytosti na 0 bude pro jakýkoliv použitý barevný tón výsledná barva pouze odstínem šedi, přičemž výsledný odstín šedi bude určen jasovou hodnotou, dále vyplývá, že při jasové hodnotě nastavené na 0 bude výsledná barva černá pro jakoukoliv kombinaci parametrů barevný tón a sytost.

Barevný rozsah HSV/HSB lze také zobrazit do 3D prostoru, na rozdíl od předchozích dvou modelů se však nezobrazuje pomocí jednotkové krychle, ale pomocí šestibokého jehlanu, obrázek 2.3, přičemž dominantní barvy (s maximální sytostí) leží na plášti jehlanu, čisté barvy na obvodu podstavy. Černá barva se zobrazuje do vrcholu jehlanu, bíla do středu podstavy.

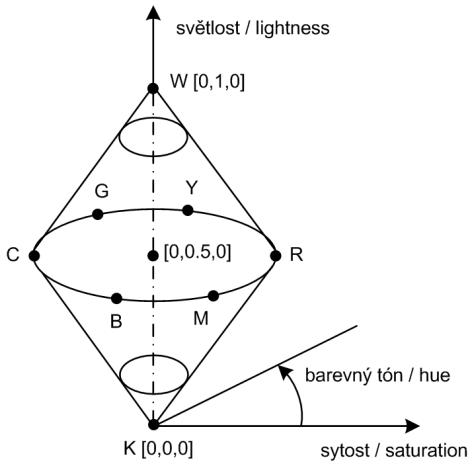


Obrázek 2.3: Geometrická reprezentace prostoru HSV, resp. HSB [1].

2.5 HLS

Model HLS staví na modelu HSV/HSB, je tvořen opět třemi hlavními parametry, barevný tón (***h**ue*), světlost (***l**ightness*) a sytost (***s**aturation*). Barevný tón má stejný význam a nabývá stejných hodnot jako v modelech HSV/HSB, světlost nabývá hodnot <0,1>, kdy 0 znamená černou barvu, a 1 bílou. Sytost má stejný význam a nabývá stejných hodnot jako v modelech HSV/HSB.
Z předchozího vyplývá, že při nastavení sytosti na 0 bude pro jakýkoliv použitý barevný tón výsledná barva pouze odstínem šedi, přičemž výsledný odstín šedi bude určen světlostí, dále vyplývá, že při světlosti nastavené na 0, resp. na 1, bude výsledná barva černá, resp. bílá, pro jakoukoliv kombinaci parametrů barevný tón a sytost.

Nevýhodami HSV/HSB byla šestistěnná podstava (výhodnější je kružnice) a nesymetričnost dle osy jasu. HLS oba tyto nedostatky odstranil a pro zobrazení do 3D prostoru používá dvojici kuželů, kdy oba tyto kužely mají společnou podstavu, obrázek 2.4, a dominantní barvy (s maximální sytostí) leží na pláštích kuželů, čisté barvy na obvodu podstavy. Černá barva se zobrazuje do vrcholu spodního kužele, bíla do vrcholu vrchního kužele.



Obrázek 2.4: Geometrická reprezentace prostoru HLS [1].

2.6 YUV

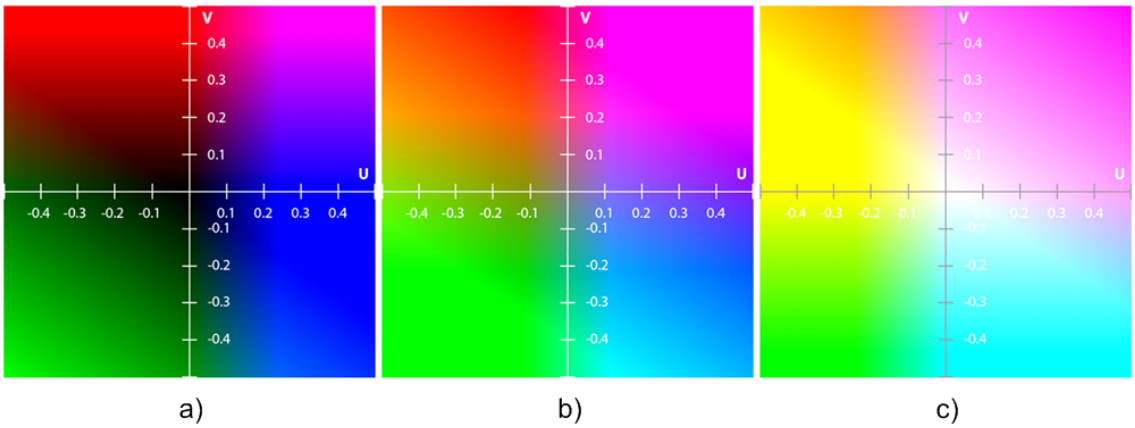
Model YUV je požíván především pro přenos televizních signálů v normě PAL a HDTV [1,3]. YUV prostor využívá tři základní složky, jednu složku obsahující informaci o jasu (**Y**) a dvě složky nesoucí barevné informace (**U** a **V**). Složka jasu nabývá hodnot <0,1>, kdy 0 značí paletu nejtmavších barevných odstínů a 1 naopak znamená maximálně světlé odstíny. Barevné složky U a V se pohybují v rozsazích <-0.5,0.5> a používají se k určení výsledné barvy podle principů patrných z obrázku 5.

Z tohoto rozdělení plyne možnost použít stejný jasový signál Y pro barevné i černobílé televizory, což je důvod, proč se tento barevný model uchytil pro přenos televizních signálů [1]. V literatuře se u tohoto barevného modelu můžeme setkat i s označením [Y,B-Y,R-Y], nebo

[Y, 0.493(B-Y), 0.877(R-Y)] [1]. Mezi důvody proč využívat YUV barevný prostor v počítačových aplikacích (detektorech) patří jednoduchý a bezztrátový převod barevného obrazu do stupňů šedi a vice versa, a možnost zpracovávat vstupní signál bez nutnosti převodu mezi barevnými prostory, jelikož snímací zařízení často dodávají na vstup signál kódovaný právě v barevném modelu YUV [8]. Pro převod z barevného modelu RGB do YUV se používá maticové násobení podle následujícího vzorce[1]:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \qquad (2.5)$$

Na následujícím obrázku jsou zobrazeny tři barevné řezy YUV barevného prostoru, pro různé hodnoty Y: obrázek 2.5-a) pro Y=0, obrázek 2.5-b) pro Y=0.5 a obrázek 2.5-c) pro Y=1, hodnoty barevných složek U a V se pohybují v rozsahu <-0.5,0.5> pro všechny tři volby Y.



Obrázek 2.5: Barevný řez YUV prostoru pro a) Y=0, b) Y=0.5 a c) Y=1 podklad pro obrázek je převzat z [30].

2.7 YCbCr, YIQ

Model YC_BC_R spadá do stejné rodiny jako barevný model YUV, na rozdíl od YUV je však používán pro přenos televizních signálů v normě SECAM (televizní signál, který byl poprvé používán ve Francii). Barevný prostor YC_BC_R je používán také v rastrovém formátu JPEG [1]. YC_BC_R prostor využívá tři základní složky, informaci o jasu (**Y**), která má stejný význam jako složka Y v rozložení YUV a dvě složky nesoucí barevné informace ($\mathbf{C_R}$ a $\mathbf{C_B}$), které mají obdobný význam jako složky U a V v modelu YUV. Dle normy CCIR-601 má hodnota Y nabývat hodnot z rozsahu <0,1> a hodnoty C_B a C_R se mají pohybovat v rozmezí <-0.5,0.5> [1]. Při celočíselné reprezentaci hodnot C_B a C_R na jednom bytu je nutné k reálné hodnotě nejprve přičíst konstantu 0.5 a takto upravenou hodnotu teprve převádět na celočíselný rozsah, reálná hodnota -0.5, resp. 0.0, resp. +0.5 potom odpovídá celočíselné hodnotě 0, resp. 128, resp. 255. Pro převod z barevného modelu RGB do YUV se používá maticové násobení podle následujícího vzorce [1]:

$$\begin{bmatrix} Y \\ C_B \\ C_R \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \qquad (2.6)$$

Posledním zástupcem, který spadá do rodiny barevných modelů začínajících Y, je barevný model YIQ, který je používán pro přenos televizních signálu v normě NTSC (norma používaná především v severní a střední Americe) [1,3].

2.8 Šedo-tónový model

Šedo-tónový model (*grayscale*) je požíván v situacích, kdy barva nepřináší žádnou dodatečnou informaci. S takovými situacemi se setkáváme např. v oblastech zpracování obrazu při detekcích založených na tvarech předmětů. Tento model využívá jediné složky, intenzity bílého světla (***i**ntensity*) a nabývá hodnot v rozsahu <0,1>, kdy 0 odpovídá černé barvě a 1 bílé barvě. Z předchozího vyplývá, že v situacích, kdy je barva bezpředmětná, má šedo-tónový model třetinové, či čtvrtinové paměťové nároky. Tento model bývá nejčastěji reprezentován jedním bytem, kdy se reálný rozsah převede na 256 hodnot, přičemž reálná hodnota 0.0, resp. 1.0 odpovídá celočíselné hodnotě 0, resp. 255. Při reprezentaci na jednom bytu, jsme tak zjevně schopni rozlišovat pouze mezi 256 odstíny šedé barvy, což nás však při vhodně zvolené převodní funkci ve většině situací nijak nelimituje. Vhodně zvolená převodní funkce zachycuje fyziologické vlastnosti lidského oka, tedy sníženou citlivost lidského oka na odstíny modré barvy a naopak zvýšenou citlivost na odstíny zelené barvy. Optimální převodní funkce z barevného modelu RGB do stupňů šedi je popsána následujícím vzorcem [1]:

$$I = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B. \qquad (2.7)$$

Není náhodou, že konstanty v tomto vzorci se přesně shodují s konstantami použitými ve vzorcích uvedenými v částech o barevných modelech YUV a YC_BC_R, funkce pro převod z těchto modelů do stupňů šedi je pak velice jednoduchá, konkrétně $I = Y$, což jinak řečeno znamená, že barevné složky *U* a *V*, resp. C_B a C_R jsou jednoduše zahozeny.

2.9 Indexovaná barva

Poslední barevný model, který stojí za zmínku je model, ve kterém se používá paleta barev. Tento model je vhodné použít v situacích, při kterých je důležitým kritériem nízká paměťová náročnost a současně je pracováno s obrazy, které neobsahují plynulé barevné přechody a naopak jsou tvořeny spíše souvislými plochami jedné barvy. Počet barev uložených v paletě reflektuje počet bitů použitých k reprezentaci barvy, např. použijeme-li reprezentaci barev na jednom bytu, dostáváme možnost adresovat až 256 barev, čili paleta by měla obsahovat 256 různých barev. Máme-li pak

obraz, či animaci, která používá méně jak 256 barev, které jsou náhodně rozloženy v barevném spektru, pak tento model přináší výhodu, jelikož jsme schopni do palety vybrat přímo oněch 256 konkrétních barev, a výsledný obraz pak není nijak zkreslen, klasické modely oproti tomu reprezentují kompletní škálu barev, které jsou lineárně odstupňované a výsledné barvy, jsou poté mapovány na nejbližší podobnou barvu z celé škály. Nevýhodou je předávání palety mezi tím, kdo obraz, či animaci vytvořil a tím kdo obraz, či animaci používá. Tento model je velice nevhodný pro reprezentaci fotografií [4].

3 Detekce lidské kůže v obraze

Cílem detektorů lidské kůže je co nejlépe rozlišovat pixely, které zobrazují lidskou kůži od těch, které lidskou kůži nezobrazují.

V této kapitole bude popsáno pouze *pixel-based* klasifikování lidské kůže, což je proces, kde klasifikace probíhá pro každý pixel zvlášť, resp. při klasifikaci pixelu nezáleží na hodnotách sousedních pixelů. Dále existuje ještě přístup *region-based* klasifikování, kam spadají metody, které při klasifikování okolní pixely zohledňují [5]. Následuje výčet metod detekování kůže, jež jsou rozděleny do kategorií dle dělení uvedeného v [5,7].

3.1 Explicitně definované metody detekce kůže

Detektory jsou založené na přesném definování hranic, resp. intervalů, při rozhodování zda pixel klasifikovat, či neklasifikovat jako kůži [5]. Pro různé barevné modely různé intervaly popisující kůži. Síla těchto detektorů spočívá v jednoduchosti rozhodování a implementace, což vede k velmi vysoké rychlosti klasifikace. Nevýhoda spočívá v určení barevného modelu a barevných rozsahů pro kůži, jež by nejlépe postihovaly aplikační oblast, tímto problém se zabývá např. [6]. Doporučené explicitní hranice pro barevný model RGB jsou dány vzorcem [5]:

$$\begin{aligned} &R > 95 \wedge G > 40 \wedge B > 20 \wedge \\ &\left|R - G\right| > 15 \wedge \\ &R > B \wedge R > G \wedge \\ &\max\{R,G,B\} - \min\{R,G,B\} > 15 \end{aligned} \quad , \tag{3.1}$$

pro barevný model YC_BC_R jsou dány vzorcem [9,10]:

$$\begin{aligned} &Y > 80 \wedge \\ &85 < C_B < 135 \wedge \\ &135 < C_R < 180 \end{aligned} \quad . \tag{3.2}$$

3.2 Neparametrické metody detekce kůže

Metody z této kategorie provádí odhad rozložení barev kůže z trénovací množiny dat bez potřeby dodávat explicitní pravidla pro detekování [5], tento odhad je zachycen pomocí takzvané pravděpodobnostní mapy, resp. pomocí takzvané binární mapy [7]. Pravděpodobnostní mapa ke každé kombinaci složek daného barevného prostoru, resp. ke každému barevnému vektoru přiřazuje pravděpodobnost s jakou daná kombinace barev, resp. daný barevný vektor odpovídá barvě lidské kůže. Binární mapa oproti tomu, ke každému barevnému vektoru striktně přiřazuje pouze hodnotu 0 nebo 1, kdy 0 znamená, že daný barevný vektor neodpovídá lidské kůži a 1 naopak, že

daný barevný vektor lidské kůži odpovídá. Konstrukce pravděpodobnostní, resp. binární mapy z trénovacích dat probíhá většinou ještě před samotným během detekce. Z předchozího textu vyplývá, že metody z této kategorie přidávají paměťové nároky, pro uložení mapy, které ale kompenzují nezávislostí na tvaru distribuční funkce a tedy nezávislostí na použitém barevném modelu [5].

3.3 Parametrické metody detekce kůže

Parametrické metody pracují pouze s těmi složkami barevného prostoru, které obsahují informace o barvě, tzn., že ignorují informace o jasu barvy. Kvalita fungování těchto metod je více závislá na použitém barevném prostoru v porovnání s neparametrickými metodami [5]. Parametrické metody mají však užitečnou schopnost interpolovat a zobecňovat neúplná trénovací data, toho dosahují nalezením matematického modelu, který nejlépe odpovídá danému podprostoru barvy kůže. Parametry matematického modelu mohou být vypočteny buď z trénovacích dat, anebo mohou být zadány explicitně na základě experimentálních výsledků, v obou případech je vhodné dbát na skutečnost, že podprostor barvy kůže má většinou eliptický tvar s přibližně normálním rozložením [7]. Většina metod spadající do této kategorie ignoruje barevné statistiky o bodech spadající mimo podprostor barvy kůže, což společně se závislostí parametrických metod na tvaru podprostoru barvy kůže vede k vysoké míře chybně pozitivně klasifikovaných vzorů v porovnání s neparametrickými metodami. Parametrické metody mají nízké paměťové nároky (je nutné reprezentovat pouze parametry matematického modelu popisujícího podprostor barvy kůže). Klasickými zástupci těchto metod detekce kůže jsou metody využívající jako matematický model jednu nebo i více Gaussových křivek. Při použití kombinace Gaussových křivek je však dosaženo velmi pomalé rychlosti, a to jak při trénování, tak i při samotném detekování, proto se jako matematický model častěji používá pouze jedna Gaussova křivka.

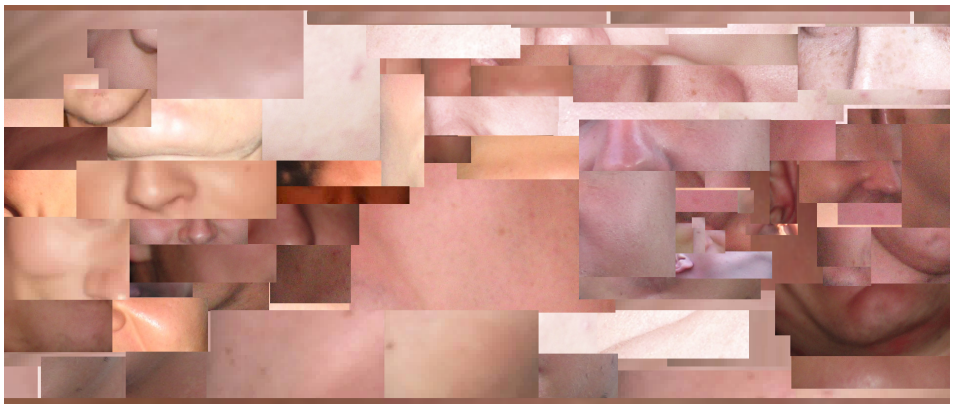
3.4 Implementace parametrického detektoru

Jelikož implementace parametrických detektorů je méně triviální v porovnání s explicitními detektory a zároveň je více používaná v porovnání s neparametrickými detektory, byla do této kapitoly zahrnuta ukázka implementace právě parametrického detektoru, konkrétně parametrického detektoru využívající jednu Gussovu křivku modelující podprostor barvy kůže. Důvodem pro popis právě této metody byla nižší závislost detekování na použitém osvětlení při srovnatelné rychlosti běhu v porovnání s explicitními metodami využívajícími barevný prostoru RGB, či YC_BC_R. Výsledná metoda využívá barevný prostor YC_BC_R, čímž je dosaženo nižších paměťových nároků, jednodušší implementace, rychlejšího rozhodovacího algoritmu při zachování srovnatelné míry úspěšnosti detekování v porovnání s používáním této metody nad barevným modelem RGB. Důvodem lepších výsledků je, že YC_BC_R obsahuje pouze dvě složky obsahující barevné informace, na rozdíl od RGB,

kde všechny tři složky jsou spojeny s barvou. Průběh tvorby detektoru kůže by se dal rozdělit do následujících kroků:

3.4.1 Tvorba trénovací množiny dat

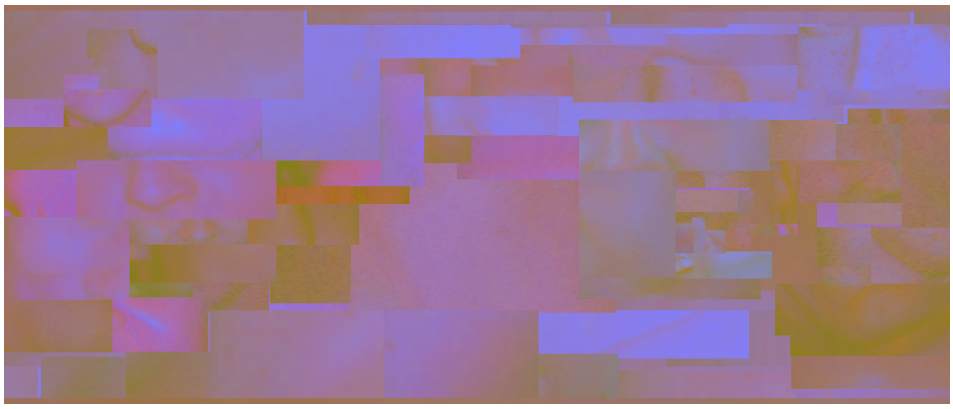
Pro úspěšnou detekci je naprosto stěžejní kvalitně sestavená množina trénovacích dat, čím více správně vybraných dat bude trénovací množina obsahovat, tím kvalitnější bude i detekce. Velikost trénovací množiny má negativní dopad na čas potřebný k natrénování detektoru, avšak rychlost samotné detekce už nijak neovlivní. Správný výběr především znamená, že v trénovacích datech by se neměly vyskytovat žádné vzory, jež by neodpovídaly hledaným datům, v našem případě lidské kůži. Trénovací množina dat použitá v této implementaci je reprezentována jedním souhrnným obrázkem, který má rozměry 1920x810 pixelů a ve své zmenšené podobě je zobrazen na obrázku 3.1. Souhrnný trénovací obrázek byl vytvořen zkombinováním mnoha výseků fotek nezahalených lidských tváří. Byla snaha v trénovacích datech postihnout různé intenzity osvětlení, různé části obličeje, různá pohlaví vzorů. Pro lepší funkci detektoru v našich končinách byly do trénovací množiny zahrnuty pouze osoby bíle pleti, z čehož plynou limitace pro používání tohoto detektoru, např. vůči osobám tmavé pleti.



Obrázek 3.1: Trénovací množina pro detekci kůže.

3.4.2 Převod trénovacích dat do YC_BC_R

Jak již bylo zmíněno, implementovaný detektor pracuje nad barevným prostorem YC_BC_R, trénovací data používají barevný prostor RGB. Z tohoto důvodu je nutné trénovací data převést do barevného prostoru YC_BC_R, podle vzorce 2.5 uvedeného v kapitole 2.7. Výsledek této fáze je presentován na obrázku 3.2. Barevný model YC_BC_R je uložen na 24 bitech, kdy každé složce přísluší 8 bitů, z čehož plyne, že složky C_B, C_R nabývají hodnot v rozsahu <0,255>, kdy 0 reprezentuje hodnotu -0.5 a 255 reprezentuje hodnotu 0.5.



Obrázek 3.2: Trénovací množina pro detekci kůže převedená do YC_BC_R.

3.4.3 Určení váženého středu trénovacích dat

Výpočet kovarianční matice prováděný v dalším kroku využívá váženého středu trénovacích dat, označovaného μ_S, proto je tento bod spočítán dopředu již v tomto kroku podle následujícího vzorce:

$$\mu_S = \left[\frac{1}{n} \sum_{y=0}^{h} \sum_{x=0}^{w} Cb_{x,y}, \frac{1}{n} \sum_{y=0}^{h} \sum_{x=0}^{w} Cr_{x,y} \right], \tag{3.3}$$

kde *h* odpovídá výšce použitého trénovacího obrázku z kroku 2, *w* odpovídá šířce stejného obrázku, *Cb* odpovídá barevné složce C_B pixelu o souřadnicích *x*, *y* v trénovacím obrázku z kroku 2, *Cr* odpovídá barevné složce C_R stejného pixelu a *n* odpovídá počtu všech pixelů v obrázku z kroku 2, resp. $n = w \cdot h$. Aplikováním vzorce 3.1 na obrázek z kroku 2 dostaneme výsledek $\mu_S = [114.5, 154.5]$.

3.4.4 Výpočet kovarianční matice z trénovacích dat

Kovarianční matice, označovaná Σ_S je matice symetrická podle hlavní diagonály, určuje natočení a šířku výsledné Gaussovy křivky a v případě barevného prostoru YC_BC_R, resp. RGB má rozměry *2x2*, resp. *3x3*. Koeficienty kovarianční matice pro barevný prostor YC_BC_R lze vypočítat podle následujícího vzorce [5]:

$$\Sigma_S = \frac{1}{n-1} \sum_{y=0}^{h} \sum_{x=0}^{w} (c_{x,y} - \mu_s)(c_{x,y} - \mu_s)^T, \qquad c_{x,y} = [Cb_{x,y}, Cr_{x,y}] \tag{3.4}$$

kde význam symbolů μ_S, *h, w, Cr, Cb* a *n* je shodný s jejich významem definovaným v kroku 3. Aplikováním vzorce 3.4 na obrázek z kroku 2 dostaneme výsledek

$$\Sigma_S = \begin{bmatrix} 72.04 & -31.17 \\ -31.17 & 31.18 \end{bmatrix}.$$

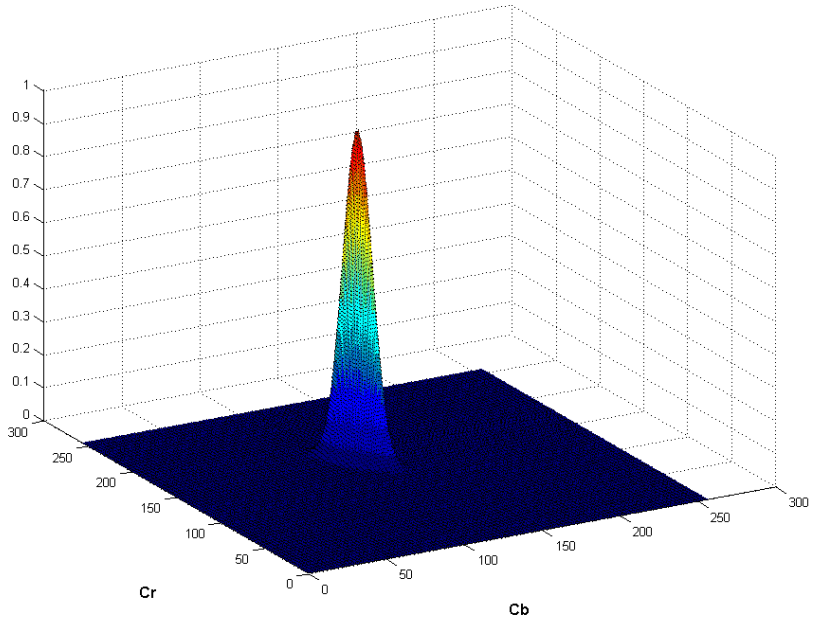
3.4.5 Vytvoření pravděpodobnostní matice

Gaussovu křivka je v implementaci reflektována při výpočtu pravděpodobnostní matice o rozměrech 256x256, kde je ke každé kombinaci barevných složek C_B a C_R přiřazena hodnota reprezentující pravděpodobnost s jakou tato kombinace spadá do podprostoru barvy kůže. Pravděpodobnost každé jednotlivé kombinace C_B, C_R se vypočte podle vzorce [5]:

$$p(Cb,Cr) = \frac{1}{2\pi\sqrt{\Sigma_s}} \cdot e^{-\frac{1}{2}(c-\mu_s)^T \Sigma_s^{-1}(c-\mu_s)}, \qquad (3.5)$$
$$c = [Cb, Cr]$$

kde význam symbolů μ_S, *h, w, Cr, Cb* a *n* je shodný s jejich významem definovaným v kroku 3 a význam symbolu Σ_S je shodný s jeho významem definovaným v korku 4.

Zobrazením pravděpodobnostní matice ve 3D, kde ose *x* odpovídá barevná složka C_B, ose *y* odpovídá barevná složka C_R a ose Z odpovídá pravděpodobnost, s jakou daná kombinace zobrazuje lidskou kůži, vznikne Gaussova křivka se středem v bodě μ_S, obrázek 3.3.



Obrázek 3.3: Zobrazení Gaussova pravděpodobnostního rozložení trénovacích dat, jež se odráží v datech pravděpodobnostní matice o rozměrech 256x256.

3.4.6 Aplikování pravděpodobnostní matice pro detekci kůže

Určením pravděpodobnostní matice skončila fáze trénování, a tím i veškeré doprovodné výpočty, jež parametrické metody mají navíc oproti explicitním metodám, z čehož plyne ekvivalentní rychlost detekování parametrických a explicitních metod. Fáze detekování lidské kůže implementovaného detektoru pro svou činnost z trénovací fáze využívá pouze pravděpodobnostní matici. Detekování

kůže se děje pro každý pixel zvlášť bez ohledu na sousední pixely, vlastní detekce začíná načtením testovaného snímku, poté následuje převod testovaného snímku do barevného modelu YC_BC_R, podle vzorce uvedeného v kapitole 2, průchod testovaným snímkem pixel po pixelu, pro každý pixel jsou extrahovány barevné složky C_B a C_R, pro danou kombinaci C_B a C_R je nalezen záznam v pravděpodobnostní matici, najitý záznam, resp. pravděpodobnost, že aktuální pixel zobrazuje kůži. Nyní tedy víme, s jakou pravděpodobností aktuální pixel zobrazuje kůži, což nám v některých situacích stačí jako výsledek, pokud ale potřebujeme binární klasifikaci pixelu je/není kůže, je tato pravděpodobnost ještě porovnána s explicitně definovaným prahem, pokud je pravděpodobnost vyšší než práh, je pixel klasifikován jako lidská kůže, v opačném případě pixel není klasifikován jako součást lidské kůže. Následující obrázek 3.4 presentuje podporovaných režimů implementovaného detektoru.

Obrázek 3.4: Levý horní (3.4-a) originální obraz, pravý horní (3.4-b) prahovaný originál, levý spodní (3.4-c) pravděpodobnostní reprezentace a pravý spodní (3.4-d) prahovaná pravděpodobnostní reprezentace.

Obrázek 3.4-a znázorňuje příklad testovacího obrázku. Obrázek 3.4-b vznikl detekcí s prahováním testovaného snímku tak, že pixely, jež byly pod explicitně zadaným prahem, v tomto případě 30%, byly nahrazeny nulovou pravděpodobností, resp. černou barvou, pixely, které byly vyšší než zvolený práh byly ponechány beze změny, tzn. odpovídají původním barvám z testovaného snímku. Obrázek 3.4-c znázorňuje testovaný snímek převedený do stupňů šedi, které reflektují výsledky provedené detekce bez prahování tak, že čím vyšší je pravděpodobnost, že pixel odpovídá lidské kůži, tím světlejší barvou je ve výsledku vykreslen, z čehož vyplývá, že pixely s nulovou pravděpodobností

jsou ve výsledku vykresleny černou barvu a pixely s maximální pravděpodobností jsou vykresleny bílou barvou. Obrázek 3.4-d vznikl spodním prahováním obrázku 3.4-c, tzn. pixely, které měly v obr. 3.6 pravděpodobnost nižší než explicitně zadaný práh, v tomto případě 30 %, byly nahrazeny nulovou pravděpodobností, resp. černou barvou, pixely, jež byly vyšší než práh, zůstaly beze změny.

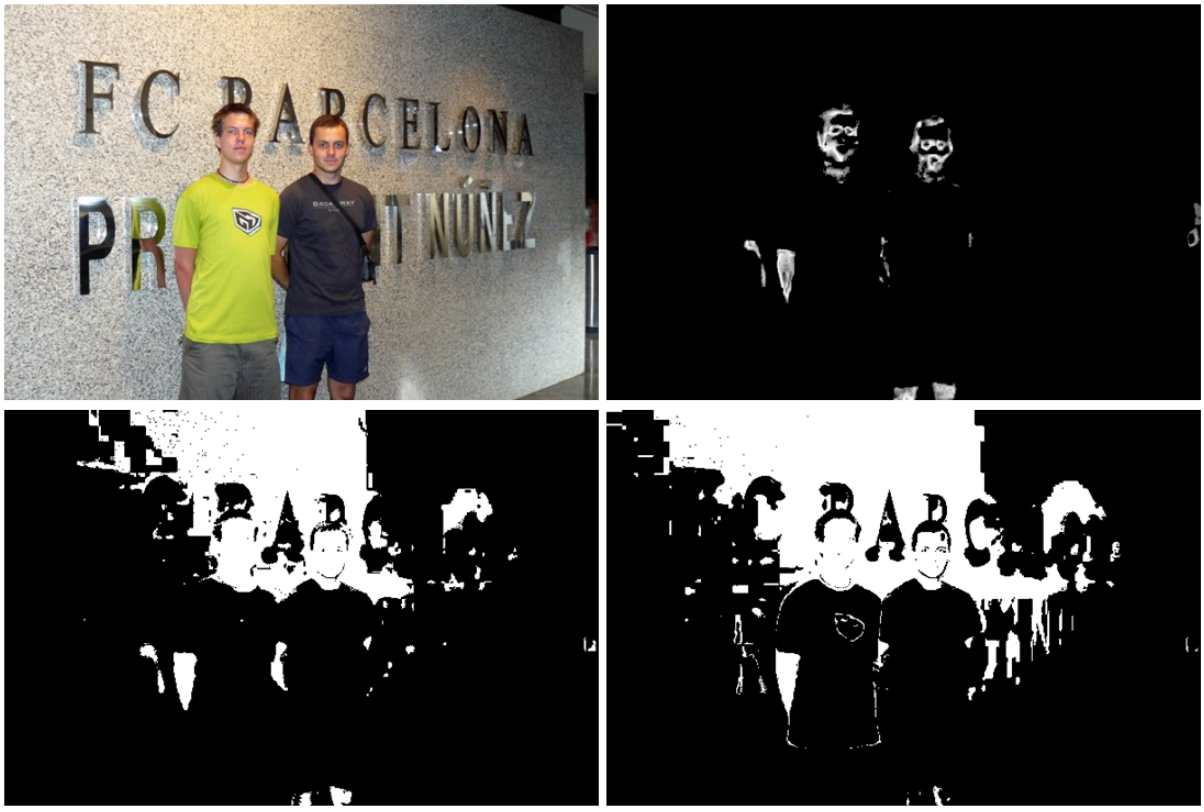
3.5 Srovnání vybraných detektorů kůže

Pro účely této práce byly zkonstruovány tři detektory kůže, explicitní detektor pracující nad barevným prostorem RGB využívající barevné intervaly definované na začátku 3. kapitoly, explicitní detektor pracující nad barevným prostorem YC_BC_R využívající barevné intervaly definované na začátku 3. kapitoly a konečně parametrický detektor využívající jednu Gaussovu křivku pro matematický model pracující nad barevným prostorem YC_BC_R. Tento detektor byl díky svým výsledkům nakonec implementován do výsledného programu, a proto byl důkladně popsán již dříve v této kapitole. Oba explicitní detektory ukázaly svou zvýšenou náchylnost na nekonstantní osvětlení v porovnání s parametrickým detektorem. Následující obrázek 3.5 presentuje výstupy všech tří detektorů pro testovaný obrázek pořízený za optimálního osvětlení:

Obrázek 3.5: Levý horní zobrazuje běžný testovaný obrázek, pravý horní zobrazuje výstup z Gaussova detektoru v režimu prahovaných stupňů šedi, viz 3.4-d, levý spodní zobrazuje výstup z RGB, a pravý spodní prezentuje výstup z YC_BC_R.

3.6 Zhodnocení detekce kůže

Zhodnocení je provedeno s ohledem na vhodnost použití jednotlivých detektorů při detekci lidských tváří. Detektory kůže jsou závislé na použitém osvětlení, bez ohledu na jejich druh (explicitní, neparametrické, parametrické), tato nevýhoda vychází z jejich přístupu ke klasifikování pixel po pixelu, bez ohledu na okolí, zastíněná kůže má totiž zcela jiný odstín oproti přímo ozářené kůži. Dalším častým problémem při detekcích lidské kůže v obraze je splynutí lidské kůže s pozadím scény, které má podobné barevné vlastnosti jako lidská kůže. Poslední zmíněný problém je zobrazen na obrázku 3.6. Detekce kůže je také závislá na barvě světelného zdroje. Použijeme-li jako světelný zdroj v místnosti pouze červenou žárovku, bude mít lidská kůže naprosto jinou barvu, než za klasického denního světla. Pokud nemáme žárovku specifické barvy, můžeme si změnu barvy lidské kůže nasimulovat třeba i ponořením kůže pod vodní hladinu, což znázorňuje obrázek 3.7. Z těchto skutečností lze soudit, že detektor obličeje založený pouze na detekci lidské kůže by nebyl vhodným výstupem této práce. Nicméně detektor lidské kůže je možné používat jako doplněk komplexnějšího detektoru obličejů ve videu např. pro vymezení zájmové oblasti komplexního detektoru, nebo např. jako dodatečné rozhodovací kritérium při nejistých výsledcích detekce.



Obrázek 3.6: Levý horní - testovaný obrázek s problematickým pozadím, pravý horní - úspěch Gaussova detektoru, levý spodní - neúspěch RGB detektoru, pravý spodní - neúspěch YC_BC_R detektoru.

Obrázek 3.7: Levý horní - testovaný obrázek se specifickým osvětlením, pravý horní - neúspěch Gaussova detektoru, levý spodní neúspěch RGB detektoru, pravý spodní - neúspěch YC_BC_R detektoru.

4 Detekce a sledování obličeje v obraze

Tato kapitola ze začátku představuje základní pojmy, se kterými se při tvorbě detektoru obličeje v obraze můžeme setkat. Dále popisuje několik vybraných metod používaných při tvorbě detektorů lidské tváře. Metody detekování obličeje v obraze mohou být rozděleny na metody detekující podle textury vstupního obrazu a na metody detekující podle tvaru detekovaného vzoru. Mezi metody detekující podle textury patří např. detektor založený na barvě kůže vycházející z principů detektoru barvy lidské kůže, viz kapitola 3, nebo detektor využívající metody LBP, viz kapitola 4.9. Mezi metody používané pro detekci na základě tvaru detekovaného vzoru patří např. detektory detekující na základě natrénovaného klasifikátoru, přičemž metody pro trénování klasifikátoru patří např. metoda AdaBoost, viz kapitola 4.5. V této kapitole vzhledem k omezenému rozsahu nebylo možné postihnout všechny možné metody pro detekci obličeje v obraze, proto byly voleny pouze takové metody, jejichž principy přímo ovlivnily návrh implementovaného detektoru, anebo byly popřípadě pro daný obor natolik důležité, že jejich vynecháním by došlo významné degradaci celé kapitoly.

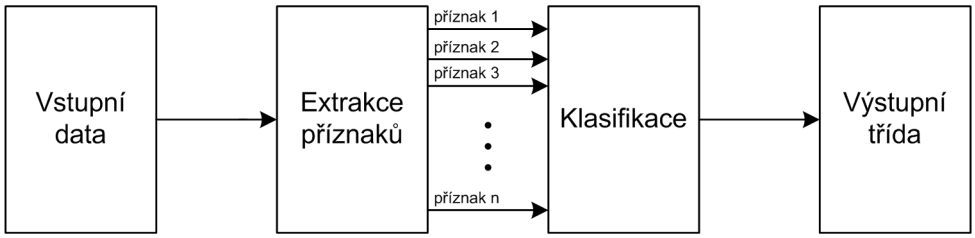
4.1 Klasifikátor

Klasifikátor je algoritmus, jenž rozděluje vstupní data do *n* různých tříd. Právě podle *n* se klasifikátory dělí na klasifikátory *binární*, kde *n=2*, a na klasifikátory vícehodnotové, kde $n > 2 \wedge n \in N$. Binární klasifikátory nejčastěji vrací hodnoty -1 pro vstupní data patřící do jedné třídy, resp. pro vstupní data neobsahující hledaný vzor, a hodnoty $+1$ pro vstupní data patřící do druhé třídy, resp. pro vstupní data obsahující hledaný vzor [13]. Vstupní data obsahující hledaný vzor bývají v literatuře často označována jako *pozitivní vstupní data* a vstupní data, jež hledaný vzor neobsahují, bývají zase často označována jako *negativní vstupní data* [25].

Vícehodnotové klasifikátory vrací *n* hodnot v určitém rozsahu, z čehož plyne, že každé třídě odpovídá jiná výstupní hodnota, jež reflektuje míru pravděpodobnosti, s jakou vstupní data z dané třídy obsahují hledaný vzor. Počáteční hodnota daného rozsahu signalizuje nulovou pravděpodobnost, poslední hodnota daného rozsahu naopak signalizuje stoprocentní pravděpodobnost, že vstupní data v této třídě obsahují hledaný vzor, jinak řečeno čím vyšší hodnotu klasifikátor pro testovaná data vrátí, tím vyšší je pravděpodobnost výskytu hledaného vzoru v těchto datech.

Do klasifikátoru jen zřídkakdy vstupují data ve své původní reprezentaci. Mnohem častěji do klasifikátoru vstupují pouze příznaky vyextrahované z původních dat, např. Haarovy příznaky, viz kapitola 4.4. Takto extrahovaných příznaků může být obecně *m*, klasifikátor pak vyhodnocuje *m* vstupů, resp. vstupní vektor extrahovaných příznaků o velikosti *m*, pro něž vrátí jeden výstup, viz obrázek 4.1. Volba příznaků, jež se budou z původních dat extrahovat, přímo ovlivňuje výslednou

funkci klasifikátoru, tudíž je velmi vhodné této volbě věnovat zvýšenou pozornost při návrhu klasifikátoru.



Obrázek 4.1: Blokové schéma procesu klasifikování.

4.2 Učení klasifikátoru

Pro správné klasifikování obecných dat klasifikátorem je nutné nejdříve tento klasifikátor naučit jak a co má klasifikovat, resp. natrénovat ho na trénovací množině dat. Kvalita trénovací množiny přímo ovlivňuje kvalitu klasifikování a tudíž i kvalitu celého detektoru, při vhodně vybíraných trénovacích datech platí vztah, že čím rozsáhlejší trénovací množina je, tím kvalitněji se klasifikátor natrénuje, ale tím více času také fáze trénovaní, resp. učení zabere. Příkladem nevhodného vybírání trénovacích dat může být proces, jehož výsledkem je trénovací množina, jež obsahuje duplicitní záznamy. Učící fáze probíhá nejčastěji pouze jednou, z čehož vyplývá nutnost nějakým způsobem uložit stav klasifikátoru, a bývá spouštěna ještě před samotným klasifikováním obecných dat. Po skončení fáze učení, se klasifikátor ve většině případů již nic nepřiučí, ale existují i varianty, kde se fáze učení a fáze detekování navzájem překrývají, z čehož plyne změna klasifikátoru s každým novým vstupem [27].

V dnešní době se používají dvě metody učení, metoda *učení bez učitele* a metoda *učení s učitelem*. Při metodě učení bez učitele jsou k trénování klasifikátoru používány vstupní data, u nichž není dopředu známo, zda se jedná o pozitivní vstupní data, či negativní vstupní data. Rozhodování, do které výstupní třídy vstupní data zařadit, pak probíhá na základě nalezených statistických zákonitostí vzorů mezi trénovacími daty. Tyto zákonitosti samotné jsou pak ale podmětem pro vznik nových statistických vzorů, jelikož je možné hledat statistické zákonitosti také ve vlivu statistických zákonitostí na řazení vstupních dat do výstupních tříd. Statistické vzory mohou být vytvářeny například na základě porovnávání charakteristických vlastností vstupních dat, extrahovaných stejnou metodou [28].

Při metodě učení s učitelem jsou při trénování klasifikátoru používány vstupní data, u nichž je předem známo, zda obsahují, resp. neobsahují hledaný vzor. Varianta učení s učitelem je v praxi mnohem častější, kromě množiny trénovacích dat lze při učení využívat ještě množinu testovacích dat, na nichž je pak možné otestovat funkci klasifikátoru a určit tak jeho chybu. Varianta učení s učitelem, kde je používána pouze trénovací množina se funkce klasifikátoru ověřuje na stejných datech, na nichž byl klasifikátor natrénován, což má jistě nižší vypovídající hodnotu v porovnání s ověřováním na odlišných datech.

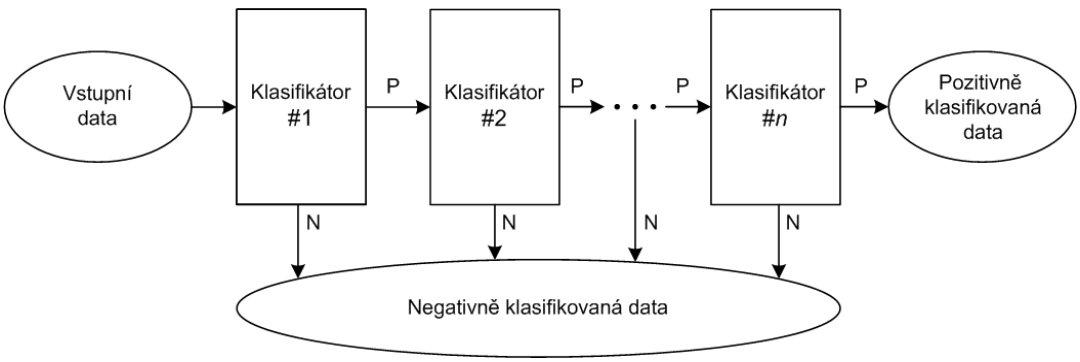
4.3 Kaskáda klasifikátorů

Kaskádové zapojení klasifikátorů je technika vycházející ze skutečnosti, že ve hledaných datech se vyskytuje často mnohem méně oblastí obsahujících hledaný vzor, než-li oblastí, jenž hledaný vzor neobsahují, např. při detekci obličeje ve fotografii se často vyskytuje mnohem více oblastí s pozadím, nežli s obličejem, a také ze skutečnosti, že lze vytvořit velmi rychlý klasifikátor, jenž do třídy pozitivně klasifikovaných dat správně zahrne téměř 100% pozitivních vstupních dat, ale navíc do této třídy chybně zahrne také 20-50% negativních vstupních dat, což má za následek, že za cenu chybné pozitivní klasifikace 20-50% negativních dat lze velmi rychle správně zamítnout 50-80% negativních vstupních dat.

Takový klasifikátor je sice velmi benevolentní, ale může klasifikovat velmi rychle, jelikož pro svou funkci potřebuje vyhodnocovat o poznání méně vstupních příznaku v porovnání s kvalitnějšími klasifikátory [12], což má dvojí dopad na rychlost celého detektoru, jelikož i blok provádějící extrahovaní příznaku, viz obrázek 4.1, pak funguje o poznání rychleji, protože ze vstupních dat extrahuje o poznání méně příznaků.

Chybně pozitivně detekovaná vstupní data lze potom odstranit předáním všech pozitivně detekovaných dat na vstup dalšího, přísnějšího, klasifikátoru k další klasifikaci, z čehož plyne, že s negativně klasifikovanými daty prvním, benevolentním, klasifikátorem se již dále nepracuje. Předávání výstupu jednoho klasifikátoru na vstup si lze představit jako zapojení výstupu jednoho klasifikátoru na vstup jiného, což odpovídá kaskádovému zapojení, od čehož je odvozen název tohoto principu.

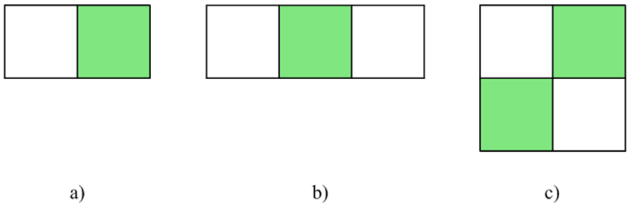
Počet takto zapojených klasifikátorů může být obecně *n*, přičemž míra tolerance k negativním vstupním datům se s narůstajícím stupněm kaskády snižuje, z čehož plyne nejvíce tolerantní a tedy i nejvíce jednoduchý klasifikátor je zapojen na začátek kaskády, a naopak nejvíce přísný a tedy i nejvíce komplexní klasifikátor je zapojen na konec kaskády. Tento přístup je výhodný, zejména proto, že časově nejnáročnější klasifikátory pracují s minimem negativních vstupních dat. Následující obrázek názorně presentuje kaskádové zapojení *n* klasifikátorů.



Obrázek 4.2: Zapojení klasifikátorů do kaskády.

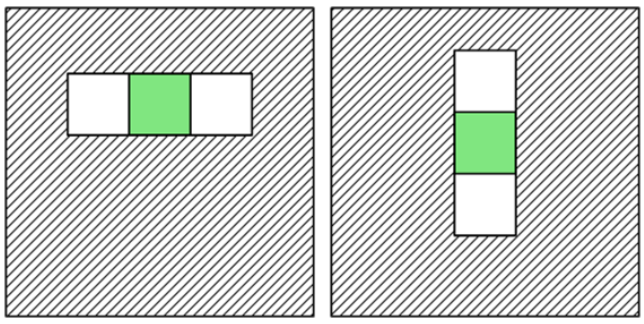
4.4 Haarovy příznaky

Haarovy příznaky jsou druhem obrazových extrahovaných příznaků, popisovaných v kapitole 4.1, ve svých strukturách odráží principy Haarovy vlnky [11]. Hodnota Haarova příznaku je počítána jako suma intenzit pixelů testovaného obrazu ležících pod světlejším úsekem, resp. světlejšími úseky Haarova příznaku, od které je odečtena suma intenzit pixelů testovaného obrazu ležících pod tmavším úsekem, resp. tmavšími úseky téhož příznaku. Haarovy příznaky se skládají ze dvou (v případě hranového Haarova příznaku), ze tří (v případě čárového Haarova příznaku), nebo ze čtyř (v případě diagonálního Haarova příznaku) obdélníkových výsečí [12]. Toto rozdělení je znázorněno na následujícím obrázku.



Obrázek 4.3: a) hranový příznak, b) čárový příznak, c) diagonální příznak.

Předešlé tři třídy Haarových příznaků jsou považovány za příznaky základní [11], kromě těchto základních existují Haarovy příznaky ze základních odvozené, které jsou označovány souhrnným názvem rozšířené Haarovy příznaky [12]. V [11] je dále presentováno řešení, kde jsou používány čtvercové Haarovy příznaky o velikosti 24x24 pixelů, přičemž velikost Haarovy vlnky je často menší, z čehož vyplývá, že nastává více možností, jak umístit Haarovu vlnku do 24x24 okna, které se mohou lišit v počátečních souřadnicích vlnky, popřípadě úhlu natočení vlnky, úhly bývají nejčastěji násobky 45°, např. na obrázku 4.4 jsou presentovány dva rozdílné způsoby umístění stejné vlnky do okna 24x24 pixelů. Výpočet hodnoty 24x24 Haarova příznaku je, stejně jako v předešlém případě, složen ze sumy intenzit pixelů pod světlejším obdélníkem, resp. světlejšími obdélníky, od které je odečtena suma intenzit pixelů pod tmavším obdélníkem, resp. tmavšími obdélníky. V tomto případě se však mohou objevit i oblasti pixelů, jež neleží ani pod světlými ani pod tmavými obdélníky, takové oblasti, na obrázku 4.4 značeny šrafováním, jsou při výpočtu ignorovány.



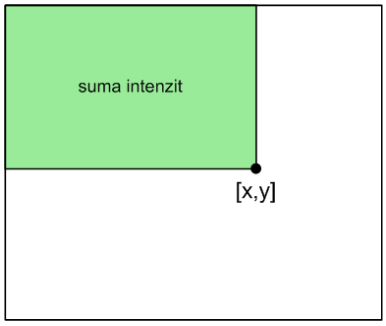
Obrázek 4.4: Umístění Haarovy vlnky do 24x24 Haarova příznaku.

4.5 Integrální obraz

Z důvodu vyhodnocování velkého množství Haarových příznaků pro jeden testovaný obraz, je často používán převod testovaného obrazu z původní podoby, nejčastěji šedo-tónový obraz, do podoby integrálního obrazu. Integrální obraz je obraz, kde se na každé souřadnici *x*, *y* nachází suma intenzit všech pixelů ležících nahoru a nalevo od pozice [*x*,*y*]. Z předchozí věty plyne, že každý integrální obraz obsahuje na pozici [0,0] hodnotu 0, a na pozici [*w*,*h*] obsahuje hodnotu určenou sumou intenzit všech pixelů v obraze, přičemž *w* odpovídá šířce celého obrazu a *h* odpovídá výšce celého obrazu. Formální zápis výpočtu integrálního obrazu lze vyjádřit následujícím vzorcem [11,12]:

$$ii(xx, yy) = \sum_{x=0}^{xx} \sum_{y=0}^{yy} i(x, y), \tag{4.1}$$

kde *xx* a *yy* odpovídají souřadnicím počítaného pixelu v integrálním obraze, *x* a *y* odpovídají souřadnicím v původní reprezentaci testovaného obrazu, *ii* je označení integrálního obrazu a i označuje původní testovaný obraz, v němž se na souřadnicích *x*, *y* nachází hodnota intenzity pixelu *x*, *y*.



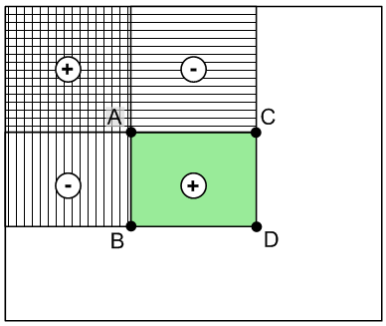
Obrázek 4.5: Příklad výpočtu hodnoty integrálního obrazu o souřadnicích *x*, *y*.

Výpočet celého integrálního obrazu je možné spočítat pouze jedním průchodem testovaného obrazu použitím postupu shrnutého následujícím vzorcem [11,12]:

$$\begin{aligned} s(-1, y) &= 0 \\ ii(x,-1) &= 0 \\ s(x, y) &= s(x-1, y) + i(x, y) \\ ii(x, y) &= ii(x, y-1) + s(x, y) \end{aligned}, \tag{4.2}$$

kde symbol *s* značí komulovaný součet hodnot[12] v řádku obrazu, v případě, že *x* adresuje sloupce a *y* adresuje řádky, veličiny *i* a *ii* mají stejný význam jako ve vzorci (4.1).

Integrálního obrazu je vhodné použít např. při výpočtu Haarových příznaků, jelikož výpočet těchto příznaků bude pak výrazně jednodušší, rychlejší a konstantní pro libovolně velké obdélníkové oblasti. Výpočet sumy intenzit jakéhokoliv obdélníkového výseku obrazu lze totiž pomocí integrálního obrazu vypočítat pomocí dvou operací sčítání a jedné operace odčítaní, názorně zobrazeno na obrázku 4.6 a formálně popsáno vzorcem (4.3).



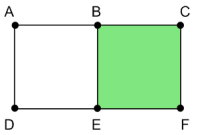
Obrázek 4.6: Příklad výpočtu integrálního obrazu pro obdélníkovou oblast určenou body *A, B, C, D*.

Na obrázku 4.6 je vidět, že oblast obdélníku určeného body A, B, C, D je dána celým obdélníkem [0,0] až D, od kterého se odečtou obdélníky [0,0] až B a [0,0] až C, čímž ale od obdélníku [0,0] až D odečítáme dvakrát oblast [0,0] až A, z čehož plyne finální přičtení obdélníku [0,0] až A. Formální zápis výpočtu sumy intenzit pod obdélníkem A, B, C, D lze popsat následujícím vzorcem:

$$\begin{aligned} \sum_{ABCD} &= ii(D) - ii(B) - ii(C) + ii(A) \\ \sum_{ABCD} &= [ii(D) + ii(A)] - [ii(B) + ii(C)] \end{aligned}, \qquad (4.3)$$

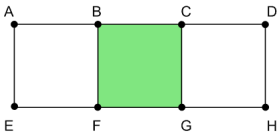
kde Σ_{ABCD} odpovídá sumě intenzit pod obdélníkem určeným body A, B, C, D, v obrázku 4.6 vyznačen zelenou barvou, rozložení bodů A, B, C, D je dáno obrázkem 4.6, význam funkce *ii* je dán vzorcem (4.1).

Jak jsme mohli vidět, Haarovy příznaky jsou často složeny z více obdélníků, které spolu však přímo sousedí, tato vlastnost se dá pomocí integrálních obrazů zohlednit úsporou počítaných bodů, názorně vysvětleno pomocí obrázku 4.7, 4.8, 4.9, kde můžeme vidět, že pro 2, resp. tři, 3. resp. 4 sousedící obdélníky je při výpočtu hodnoty Haarova příznaku, na obrázcích značena jako *S*, nutné počítat s 6-ti, resp. s 8-mi, resp. s 9-ti body, vypozorováno z [11,13].



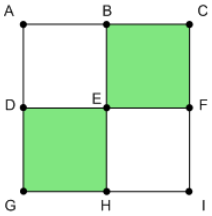
$$S = (E - B - D + A) - (F - C - E + B)$$
$$S = 2E - 2B - D + A - F + C$$
$$S = (A + C + 2E) - (2B + D + F)$$

Obrázek 4.7: Redukce výpočtů v případě dvou přímo sousedících oblastí Haarova příznaku.



$$S = [(F - B - E + A) + (H - D - G + C)] - (G - C - F + B)$$
$$S = 2F - 2B - E + A + H - D - 2G + 2C$$
$$S = (A + 2C + 2F + H) - (2B + D + E + 2G)$$

Obrázek 4.8: Redukce výpočtů v případě tří přímo sousedících oblastí Haarova příznaku.



$$S = [(E - B - D + A) + (I - F - H + E)] - [(F - C - E + B) + (H - E - G + D)]$$
$$S = 4E - 2B - 2D + A + I - 2F + C - 2H + G$$
$$S = (A + C + 4E + G + I) - (2B + 2D + 2F + 2H)$$

Obrázek 4.9: Redukce výpočtů v případě čtyř přímo sousedících oblastí Haarova příznaku.

Ze všech zmíněných vlastností integrálních obrazů jasně vyplývá, že jejich použitím pro výpočty Haarových příznaků můžeme dosáhnout vyšší rychlosti vyhodnocování a především nezávislosti výpočtu na velikosti porovnávané oblasti, ve srovnání s metodou, používající pro ten samý výpočet klasický přístup sčítání intenzit pixel po pixelu.

4.6 AdaBoost

AdaBoost (z anglického *Adaptive boosting*) je algoritmem strojového učení s učitelem, je používán k natrénování klasifikátoru vhodného k rozpoznávání vzorů v obraze, např. detekce lidského obličeje. Na rozdíl od dříve zmíněného detektoru lidské tváře založeného na klasifikaci podle barvy lidské kůže, AdaBoost produkuje klasifikátor, jenž v obraze vyhledává vzory naučené při trénování, z čehož plyne jak nezávislost na barevném modelu testovaného snímku, tak i nezávislost na použitém osvětlení. AdaBoost prohledává velkou, často ohromnou (jeho největší převaha oproti konkurenci je v oblastech s 1000 až 10 000 000 vzorky [13]), databázi slabých klasifikátorů *H*, ze kterých vybírá ty, které na trénovacích datech *S* dosahují nejlepších výsledků klasifikace a lineární kombinací takto vybraných slabých klasifikátorů, vzniká jeden výsledný nelineární silný klasifikátor *H(x)*. Pod pojmem slabý klasifikátor, označovaného $h_t(x)$, si můžeme představit jednoduchou funkci, která trénovací data klasifikuje do dvou skupin, obsahuje / neobsahuje hledaný objekt, resp. *+1* / *−1*, pouze na základě hodnoty jednoho příznaku (např. Haarova příznaku) a hodnoty prahu. Na slabý klasifikátor je kladen jediný požadavek, jeho úspěšnost musí být vyšší než 50%, tzn. musí trénovací data klasifikovat lépe, než náhodná klasifikaci, z čehož plyne, že při úspěšnosti nižší než 50% stačí ke splnění podmínky daný klasifikátor negovat. Trénovací množina *S* je tvořena dvojicemi *(x,y)*, kde *x* je výsek trénovacího obrazu (např. 24x24 pixelů) a hodnota y signalizuje, že trénovací úsek *x* obsahuje hledaný vzor v případě $y = +1$, resp. neobsahuje hledaný vzor v případě $y = -1$. Zmíněnou konstrukci silného klasifikátoru provádí AdaBoost iterativním způsobem, přičemž v každé iteraci urychluje hledání vážením trénovací množiny váhami D_t [12], kdy váhy jednotlivých položek, resp. dvojic (x,y) trénovací množiny jsou na začátku inicializovány rovnoměrně, konkrétně na hodnotu $\frac{1}{m}$, kde *m* odpovídá počtu záznamů v trénovací množině dat, a v každé další iteraci jsou upřednostňovány dvojice (x,y) s nejhoršími výsledky, což vede k rychlejšímu nalezení vhodných slabých klasifikátorů pro klasifikování právě těchto problémových položek a k redukci trénovací chyby exponenciálně v závislosti na rostoucím počtu slabých klasifikátorů [11]. Použití vysokého počtu klasifikátorů může vést k přetrénování výsledného klasifikátoru, což znamená, že klasifikátor bezchybně klasifikuje trénovací data, ale při použití na obecná data selhává, jelikož je pro obecná data až moc konkrétní. Nutno poznamenat, že k přetrénování dochází u AdaBoostu jen velmi zřídka [13] a nejčastěji je to způsobeno vlivem málo obsáhlé trénovací množiny dat. Nevýhodou AdaBoostu je absolutní důvěra ve všechna trénovací data, což odstraňují některé modifikace AdaBoostu, jež

vyžadující explicitní nastavení míry spolehlivosti dat, což vede k větší odolnosti vůči šumu [13]. Následuje formální zápis algoritmu převzatý z [11,12]:

Vstup:

Trénovací množina: $S = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$

Množina slabých klasifikátorů *H*

Počet iterací *T*

Inicializace:

Pro všechny položky z trénovací množiny *S* vypočítej počáteční váhy $D_1(i)$ dle vzorce:

$$D_1(i) = \frac{1}{m}, \text{ kde } i \in N \wedge 1 \leq i \leq m \tag{4.4}$$

Cyklus pro *t = 1, …, T:*

1. Pro všechny slabé klasifikátory h_j z množiny *H* vypočítej váženou trénovací chybu dle vzorce:

$$\varepsilon_j = \sum_{i=1}^{m} D_t(i) \cdot [h_j(x_i) \neq y_i] \tag{4.5}$$

2. Z množiny *H* vyber aktuální slabý klasifikátor h_t dle vzorce:

$$h_t = \arg\min_{h_j \in H} \varepsilon_j \tag{4.6}$$

3. Pokud je chyba aktuálního slabého klasifikátoru ε_t rovna 0 nebo pokud je $\varepsilon_t \geq 0.5$, pak konec cyklu

4. Nastav váhu aktuálního slabého klasifikátoru α_t dle vzorce:

$$\alpha_t = 0.5 \cdot \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \tag{4.7}$$

5. Pro všechny položky z trénovací množiny *S* vypočítej nové váhy $D_{t+1}(i)$ dle vzorce:

$$D_{t+1}(i) = \frac{D_t(i) \cdot e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$
$$Z_t = \sum_{i=1}^{m} D_t(i) \cdot e^{-\alpha_t y_i h_t(x_i)} \tag{4.8}$$

Výstup:

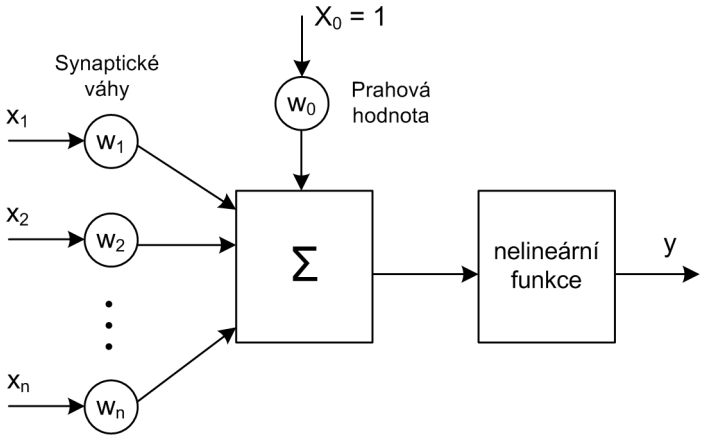
Silný klasifikátor *H(x)* určený dle vzorce:

$$H(x) = sign\left[\sum_{t=1}^{T} \alpha_t \cdot h_t(x)\right] \tag{4.9}$$

Algoritmus 4.10: AdaBoost [11, 12].

4.7 Neuronové sítě

Dalším mechanismem použitým k detekci obličeje v obraze mohou být umělé neuronové sítě, kde je snaha uměle napodobit základní principy biologické neuronové sítě lidského mozku. V dalším textu bude o umělých neuronových sítích mluveno již jen zkráceně jako o neuronových sítích. Neuronové sítě, jak již sám název napovídá, jsou tvořeny neurony. Matematický model pro neuron vytvořil W. McCulloch a W. Pitts [16]. Neuron využívající tohoto matematického modelu je složen z vstupní, výstupní a funkční části, viz obrázek 4.10, který byl vytvořen na základě [14,15,19].



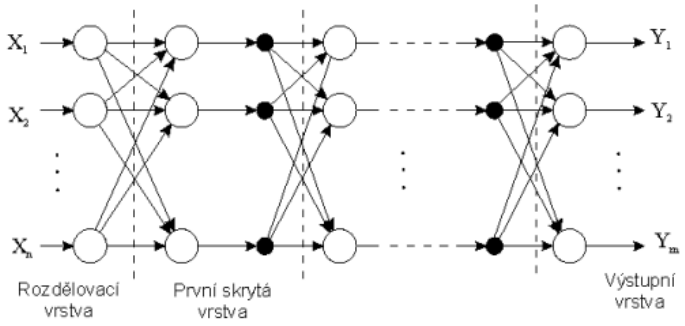
Obrázek 4.11: Blokové schéma jednoho neuronu.

Vstupní část je tvořena ze vstupů a jim přiřazených synaptických vah. Váhové koeficienty v této části v podstatě realizují paměť neuronu. Tato část navíc obsahuje jeden speciální vstup x_o, na nějž není připojen výstup žádného jiného neuronu, ale je použit k zavedení konstantní veličiny do neuronu. Tato konstanta pak funguje jako práh, resp. aktivační práh [15], nepřekročí-li suma vážených vstupů tuto konstantu, nedojde k aktivaci funkční části a výstup tak zůstane beze změny, z čehož plyne, že výstup se nemusí měnit s každou změnou vstupu. Funkční část je obecně tvořena jednoduchou nelineární funkcí, jež na základě synaptických vah a odpovídajících vstupů, získaných ze vstupní části, vygeneruje odpovídající výstup. Výstupní část neuronu se stará o předání výstupního signálu na patřičné vstupy dalších neuronů [14]. Matematickou definici neuronu uvádí následující vzorec [15]:

$$y = f\left(\sum w_i \cdot x_i + \Theta\right) \qquad (4.10),$$

kde x_i odpovídá hodnotě na vstupu i, w_i odpovídá synaptické váze vstupu x_i, Θ odpovídá hodnotě aktivačního prahu, n odpovídá celkovému počtu vstupů, f odpovídá obecné nelineární funkci a y odpovídá hodnotě výstupu.

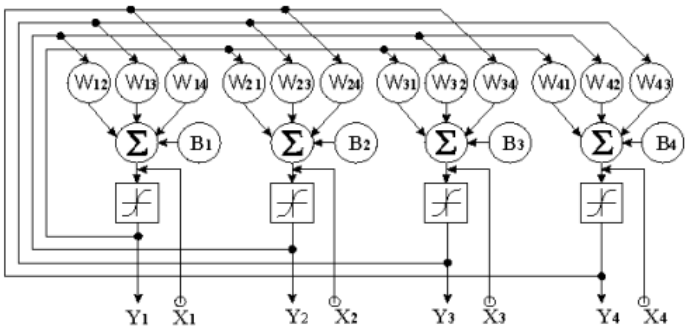
Neurony v neuronových sítích bývají často sdružovány do vrstev, kde výstupy neuronů z jedné vrstvy bývají přiváděny na vstupy obecně každého neuronu z bezprostředně následující vrstvy, obrázek 4.11.



Obrázek 4.12: Názorná ukázka dělení neuronové sítě do tří typů vrstev [14].

První vrstva neuronové sítě je pak nazývána vrstvou vstupní, poslední vrstva se nazývá výstupní a ostatní vrstvy, jež leží mezi vstupní a výstupní vrstvou, se nazývají skryté. Vstupní vrstva přijímá všechny vstupy nutné pro fungování celé neuronové sítě a deleguje je na vstupy všech neuronů z bezprostředně následující vrstvy. Výstupní signály z neuronů začleněných ve výstupní vrstvě reflektují odezvu celé neuronové sítě pro dané vstupy. Složitost neuronové sítě se odráží na počtu skrytých vrstev, čím složitější funkci neuronová síť realizuje, tím více skrytých vrstev bude neuronová síť obsahovat [14].

Neuronové sítě lze dle [14,15] rozdělit na neuronové sítě s dopředním šířením signálu a na neuronové sítě se zpětnou vazbou, kdy používanější variantou jsou sítě s dopředním šířením [14]. Neuronová síť znázorněná na obrázku 4.11, jež je blíže popsána v předešlém odstavci, spadá do kategorie neuronových sítí s dopředním šířením signálu a přesně vykazuje charakteristické rysy této varianty sítí. Varianta sítí se zpětnou vazbou v porovnání s předchozí variantou navíc nabízí posílání výstupních signálů v rámci jedné vrstvy, resp. výstupní signály neutronů z vrstvy *n* mohou být posílány jak na vstupy neutronů z vrstvy *n+1*, tak navíc i na vstupy neutronů z vrstvy *n*, což těmto sítím umožňuje realizaci iteračních procesů, jež se používají např. při optimalizačních činnostech. Příkladem neuronové sítě se zpětnou vazbou je Hopfieldova síť, kde jsou všechny neurony propojeny navzájem každý s každým. Hopfieldova síť složenou ze čtyř neuronů znázorňuje následující obrázek.



Obrázek 4.13: Hopfieldova síť [14].

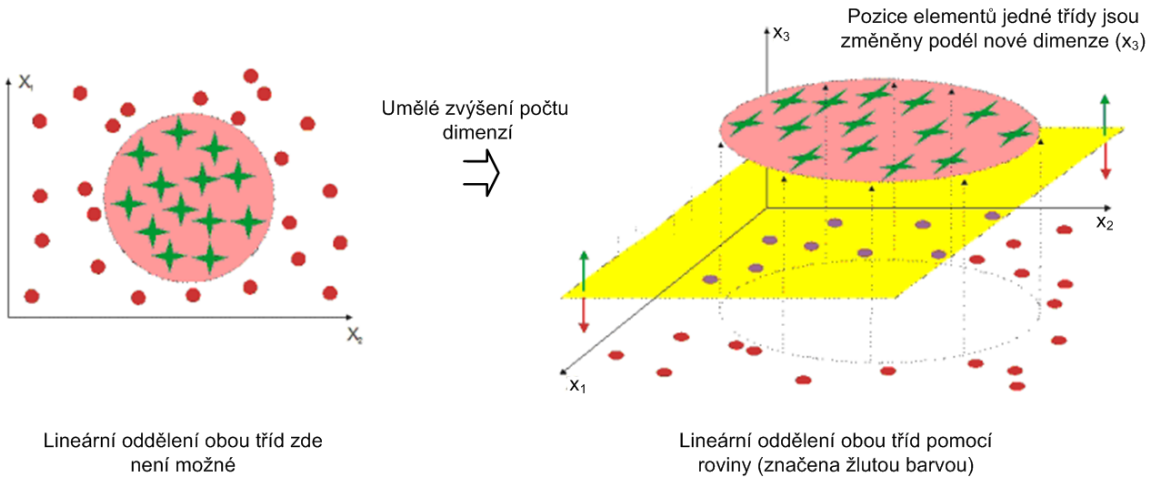
Hlavní výhodou neuronových sítí s dopředním šířením signálu je jejich schopnost se učit. Učení je důležitou fází, která předchází fázi vlastního fungování neuronové sítě. Ve fázi učení dochází k určování synaptických vah k jednotlivým vstupům tak, aby neuronová síť prováděla svojí funkci s co nejlepšími výsledky. Učení může probíhat s učitelem nebo bez učitele, viz kapitola 4.2.

Neuronové sítě mají velmi širokou aplikační oblast. Z pohledu této práce je však významné, že je lze použít také k detekci lidských obličejů v obraze, kde bývá nejčastěji používáno metody zpětného šíření chyby (*back-propagation*) [14]. Další výhodou obecně všech neuronových sítí je jistá odolnost vůči poruchám neutronu, kdy konec neutronu nezpůsobí pád celé neutronové sítě a tedy ani pád celého detektoru, může pouze dojít k jistému zhoršení fungování celé neuronové sítě. Neuronové sítě s dopředním šířením signálů lze ještě rozdělit na lineární a nelineární [14], kde rozlišujícím faktorem je linearita funkce realizované celou neuronovou sítí, z čehož plyne, že linearita funkce celé neuronové sítě se může lišit od linearity funkce realizované jedním neuronem, jež je obecně nelineární, jinými slovy, v neuronové síti, realizující lineární funkci lze použít neurony, jejichž funkce je nelineární.

4.8 SVM

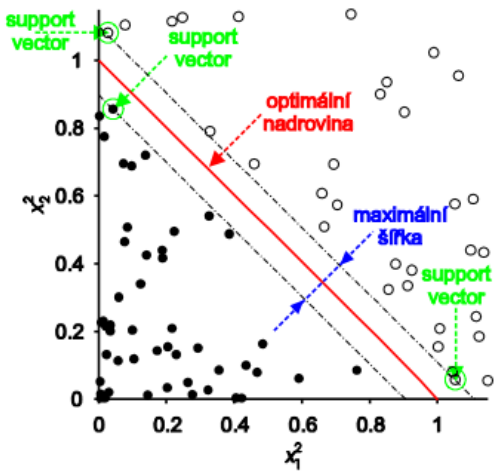
Nejjednodušším modelem neuronové sítě s dopředním šířením signálu je síť tvořená pouze jedním neuronem, taková neuronová síť je nazývána *perceptron* [19]. Výhodou perceptronu je, že ho lze učit jednoduchými a efektivními algoritmy. Nevýhoda perceptronu se však skrývá ve skutečnosti, že ho lze aplikovat pouze na klasifikaci lineárně oddělených tříd, což je velmi silné omezení pro klasifikátor. K odstranění této limitace může být použita např. metoda podpůrných vektorů (***s**upport **v**ector **m**achines*, zkráceně SVM).

Metoda podpůrných vektorů, dále již jen SVM, v případě vstupních dat patřících do tříd, jež není možné lineárně rozdělit, převede vstupní prostor do takového prostoru, kde lineární oddělení těchto tříd možné bude, takový prostor mývá často vyšší dimenzi, než původní vstupní prostor. Princip takového převodu je znázorněn na obrázku 4.13, kde je presentováno konkrétně převedení dvourozměrného prostoru do trojrozměrného prostoru, čímž je dosaženo možnosti lineárního oddělení dvou tříd pomocí roviny, přičemž v původním prostoru byly tyto dvě třídy odděleny nelineárně pomocí kružnice.



Obrázek 4.14: Princip lineárního oddělení dvou původně nelineárně oddělených rovin [18].

Po přidání dimenze je důležité rozhodnou, kam nejvhodněji umístit novou, *lineární*, hranici. V [18] je uvedeno, že otázka nalezení nejlepší pozice pro lineární hranici je řešitelná a dále je uvedeno, že principem přidávání dimenzí lze, při dostatečném počtu přidaných dimenzí, nalézt lineární oddělovač pro jakákoliv vstupní data. Problémem spojeným s používání prostorů vysokých dimenzí je nutnost reprezentace dat v takových prostorech vysokým počtem parametrů, jelikož v *n*-rozměrném prostoru jsou data reprezentovány *n* parametry, z čehož vyplývá, že i nalezený lineární oddělovač bude muset být reprezentován vysokým počtem parametrů, což vede k nízké schopnosti generalizace dat, resp. k přetrénování. Lineární oddělovač umístěný v nejvhodnější pozici, dále jen optimální oddělovač, by měl být umístěn tak, aby mezi třídami vytvořil co možná největší odstup. Při hledání optimálního oddělovače je možné přejít na *duální úlohu* [15], jejímž řešením je vektor vah, získaný váhováním trénovacích dat, podrobněji popsáno v [15,18]. Vstupní data, jimž odpovídá nulová hodnota ve vektoru vah hrají při hledání optimálního oddělovače minimální roli. Naopak vstupní data, jimž ve vstupním vektoru odpovídají nejvyšší hodnoty, jsou nazývána *supported vectors*, volně přeloženo *podpůrné vektory*, taková data leží na okraji dělící oblasti a přímo tak ovlivňují optimální pozici hledaného oddělovače. Vhledem k postupu při určování vektoru vah platí, že pouze *podpůrné vektory* jsou reprezentovány jinou, než nulovou hodnotou, z čehož plyne, že při hledání optimálního oddělovače je počítáno pouze s *podpůrnými vektory*, což vede ke zvýšení efektivity hledání. Pro lepší pochopení předešlých pojmů je uveden obrázek 4.14, na kterém jsou tyto pojmy znázorněny v situaci, kde je hledán optimální oddělovač pouze dvou tříd, tudíž v situaci pracující s binární klasifikací.



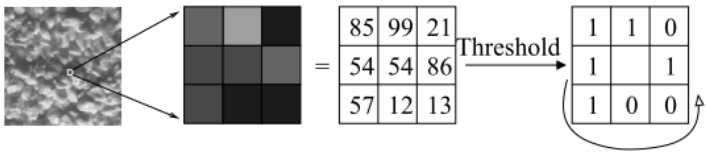
Obrázek 4.15: Znázornění pojmů *support vector*, *maximální šířka* a *optimální nadrovina* [18].

Metody využívající SVM jsou vhodné v situacích, kdy je trénování prováděno na datech, které obsahují mnohem větší poměr negativních vzorů v porovnání s pozitivními vzory, v situacích kdy je pracováno se vstupními daty reprezentovanými v prostoru s vysokým počtem dimenzí a také

v situacích, kdy jsou data ve vstupním prostoru rozeseta velmi řídce. Tyto metody naopak není vhodné používat v situacích, kdy jsou vstupní data rozdělena do vzájemně se překrývajících tříd, v takových případech je nutné zavést dodatečné proměnné, *slack variables* [15].

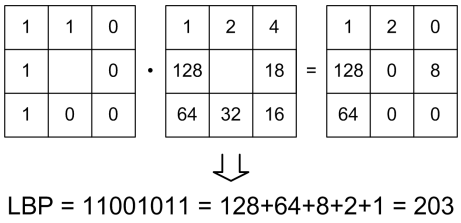
4.9 LBP

Algoritmus využívající LBP (***l**ocal **b**inary **p**attern*) pracuje přímo s hodnotami pixelů testovaného obrazu [20]. Algoritmus využívající LBP pak při vyhodnocování jednotlivých pixelů v porovnání s detektory kůže nepracuje pouze s hodnotami aktuálního pixelu, ale i s blízkým okolím vyhodnocovaného pixelu. Nejčastěji je při vyhodnocování počítáno s devíti body, aktuálním pixelem a jeho osmikolím. Na vstup LBP detektoru jsou přiváděny vstupní snímky převedené do stupňů šedi, kde každému pixelu přísluší jeden byte, což nám poskytuje rozsah 256 stupňů šedi, bližší popis této obrazové reprezentace viz kapitola 2.8, ze vstupního snímku je pak vypočtena LBP reprezentace obrazu. Pro výpočet hodnot jednotlivých pixelů v LBP reprezentaci obrazu je vždy použita matice vstupních bodů o rozměrech 3x3, jež má střed v aktuálním pixelu a zbylé položky analogicky odpovídají osmikolí aktuálního pixelu, viz obrázek 4.15, přičemž v případě výpočtu hraničních bodů jsou položky osmikolí zasahující mimo vstupní obraz vyplněny nulami. Tato matice je dále binárně prahována tak, že pokud je hodnota pixelu osmikolí nižší než hodnota středového pixelu je na místo této hodnoty v matici uložena 0, v opačném případě je na takové místo uložena 1, viz obrázek 4.15, na místo středové položky je pokaždé uložena 0.



Obrázek 4.16: Příklad prahování vstupních dat [23].

Binárně prahovaná matice je dále vynásobena váhovací maticí, viz obrázek 4.16, čímž vznikne výsledná matice. Sečtením všech hodnot ve výsledné matici dostaneme hodnotu odpovídající LBP příznaku, resp. hodnotu, jakou bude aktuální pixel reprezentován v LBP obrazu.



Obrázek 4.17: Příklad výpočtu jednoho LBP pixelu vynásobením prahovací matice váhovací maticí a následným sečtením všech hodnot výsledné matice.

Pro lepší pochopení je uveden ještě matematický postup pro výpočet hodnot výsledných pixelů LBP obrazu:

$$p\_LBP_{i,j} = \sum_{ii=0}^{3}\sum_{jj=0}^{3} m\_výsledná_{ii,jj}\ , \qquad (4.11)$$

kde $p\_LBP_{i,j}$ značí výslednou hodnotu pixelu v LBP obrazu o souřadnicích *i, j* a $m\_výsledná_{ii,jj}$ značí jednotlivé položky výsledné matice. Výsledná matice *m_výsledná* je dána následujícím vzorcem:

$$m\_vysledna = m\_prahovací \cdot m\_váhovací \qquad (4.12),$$

kde *m_váhovací* značí váhovací matici, jež je naplněna hodnotami dle vzorce (4.13):

$$m\_váhovací = \begin{bmatrix} 1 & 2 & 4 \\ 128 & 0 & 8 \\ 64 & 32 & 16 \end{bmatrix}, \qquad (4.13)$$

a *m_prahovací* označuje prahovací matici. Výpočet prahovací matice *m_prahovací* je řízen vzorcem:

$$m\_prahovací = \begin{bmatrix} f(m\_in_{0,0}) & f(m\_in_{0,1}) & f(m\_in_{0,2}) \\ f(m\_in_{1.0}) & 0 & f(m\_in_{1.2}) \\ f(m\_in_{2.0}) & f(m\_in_{2.1}) & f(m\_in_{2.2}) \end{bmatrix}, \qquad (4.14)$$

kde *m_in* označuje vstupní matici, jež je naplněna podle vzorce:

$$m\_in = \begin{bmatrix} g(-1,-1) & g(-1,0) & g(-1,1) \\ g(0,-1) & g(0,0) & g(0,1) \\ g(1,-1) & g(1,0) & g(1,1) \end{bmatrix}, \qquad (4.15)$$

a *f* označuje funkci danou vzorcem:

$$f(x) = \begin{cases} 1, x \geq m\_in_{1,1} \\ 0, x < m\_in_{1,1} \end{cases}, \qquad (4.16)$$

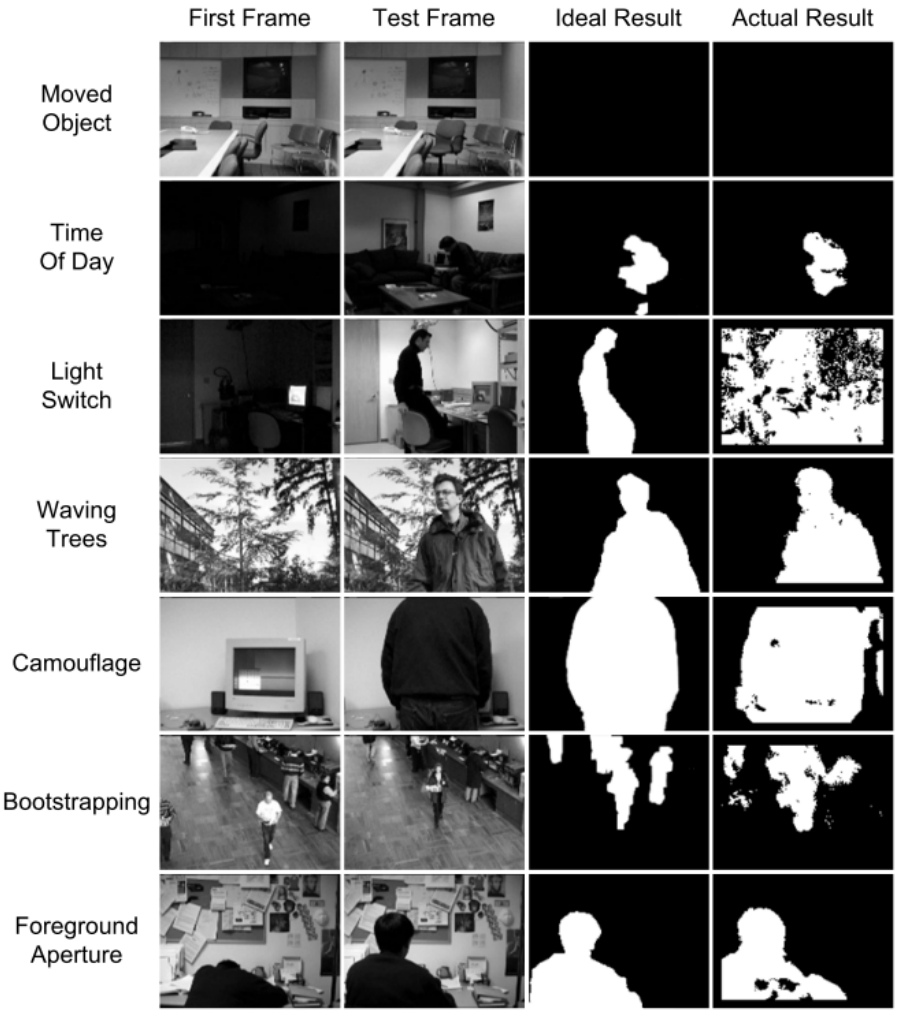
kde $m\_in_{1,1}$ označuje středovou položku již dříve definované vstupní matice *m_in*. Funkce *g* použitá ve vzorci 4.15 je definovaná jako:

$$g(x,y) = \begin{cases} p\_in_{i+x,\,j+y}, & 0 \leq (i+x) < w \wedge 0 \leq (j+y) < h \\ 0, & jinak \end{cases}, \qquad (4.17)$$

kde *p_in* odpovídá hodnotě pixelu šedo-tónového vstupního obrazu o souřadnicích *i+x, j+y*, přičemž hodnoty proměnných *i, j* přesně odpovídají jejich hodnotám ve vzorci 4.11, a konečně *w* a *h* označují rozměry, vstupního šedo-tónového, resp. výstupního LBP obrazu.

Výhodou LBP reprezentace obrazu je vysoká míra invariantnosti vůči změně osvětlení[23], jinak řečeno, při změně osvětlení celé snímané scény, např. vlivem venkovních podmínek, zůstává LBP reprezentace takřka beze změny, z čehož lze využít např. k detekci pohybu, kdy mohou být vhodně porovnávány histogramy LBP obrazu a překročení explicitně nastavené odchylky mezi dvěma po sobě jdoucími LBP obrazy musí signalizovat pohyb ve scéně, jelikož při invariantnosti vůči změně osvětlení pak jedinou příčinou vzniku této odchylky musí být právě změna scény. LBP metoda

dokáže kvalitně fungovat i při snímání venkovní scény, kde dochází k drobným změnám scény takřka pořád, stébla trávy, listí na stromech apod., tyto změny se však díky skutečnosti, že jednotlivé pixely v LBP obraze jsou počítány z osmikolí, daří velmi dobře eliminovat [24]. Příklad fungování detekce pohybu založeného na principech LBP je presentován na obrázku 4.17.



Obrázek 4.18: Ukázka detektoru pohybu používajícího metod LBP [24].

4.10 Kalmanův filtr

Tato je jediná podkapitola ze čtvrté kapitoly, jenž nepředstavuje mechanismus spojený s detekcí tváří ve videu. Tato kapitola naopak představuje Kalmanův filtr, jenž je možné použít při sledování tváří ve videu. Kalmanův filtr je rekurzivní dvoustavový filtr, kdy v každé jedné iteraci probíhá *předpověď* a *korekce* této předpovědi [33,34].

Předpověď provádí odhad souřadnic ve snímku *n* pohybujícího se objektu, ze snímku *n-1*, kde *n* je libovolné celé kladné číslo. Pokud se pak obličej pohybuje s např. konstantní rychlostí, resp. konstantním zrychlením, je možné provést odhad jeho souřadnic ve snímku *n* založenou na jeho

souřadnicích ve snímku *n-1*, použitím fyzikální rovnice pro rychlost, resp. pro zrychlení pohybujícího se předmětu.

Korekce pak po získání měřené polohy pohybujícího se objektu ve snímku *n* zpětně přizpůsobuje předešlý odhad podle této změřené polohy.

Kalmanův filtr při své definici využívá veličiny $\boldsymbol{x_t}$, $\boldsymbol{z_t}$, $\boldsymbol{P_t}$, $\boldsymbol{F}$, $\boldsymbol{H}$, $\boldsymbol{Q}$, $\boldsymbol{R}$, kde $\boldsymbol{x_t}$ je aktuální stavový vektor v čase *t*, $\boldsymbol{z_t}$ je změřená poloha objektu v čase *t*, $\boldsymbol{P_t}$ udává přesnost odhadnutého stavového vektoru x_t v čase *t*, $\boldsymbol{F}$ udává ideální pohyb objektu bez šumu, $\boldsymbol{H}$ udává převod stavového vektoru x_t na měřenou polohu z_t, **Q** udává šum vzniklý nepřesností Kalmanova filtru, *R* udává šum vzniklý měřením hodnot a $\boldsymbol{R}$ udávají nepřesnost měřené polohy z_t. Je vhodné poznamenat, že součástí Kalmanova filtru jsou ještě veličiny *B* a u, jež slouží jako kontrolní parametry, z principu Kalmanova filtru, však plyne, že tyto veličiny mohou být ignorovány v případě sledování objektů, resp. obličejů [33, 34, 35]. Kalmanův filtr pro libovolný systém pak lze popsat pomocí sedmi obecných rovnic, které jsou uvedeny v [33, 34, 35]. V této podkapitole jsou uvedeny tytéž rovnice, jež jsou však přizpůsobeny přímo systému sledování objektů, resp. obličejů ve videu. Předchozí věta konkrétně znamená, že v porovnání s obecnými rovnicemi, uvedené rovnice ignorují veličiny *B* a *u*. Následuje výčet těchto sedmi zjednodušených rovnic:

1. rovnice pro výpočet odhadnutého stavového vektoru $\overline{x_{t|t-1}}$:

$$\overline{x_{t|t-1}} = F_t \cdot \overline{x_{t-1|t-1}} \tag{4.18}$$

2. rovnice pro zisk kovarianční matice odhadu $P_{t|t-1}$:

$$P_{t|t-1} = F_t \cdot P_{t-1|t-1} \cdot F_t^T + Q_t \tag{4.19}$$

3. rovnice pro výpočet korekčního rezidua $\overline{y_t}$:

$$\overline{y_t} = z_t - H_t \cdot \overline{x_{t|t-1}} \tag{4.20}$$

4. rovnice pro zisk kovarianční matice korekce S_t:

$$S_t = H_t \cdot P_{t|t-1} \cdot H_t^T + R_t \tag{4.21}$$

5. rovnice pro výpočet optimálního Kalmanova zesílení K_t:

$$K_t = P_{t|t-1} \cdot H_t^T \cdot S_t^{-1} \tag{4.22}$$

6. rovnice pro výpočet zpětného odhadu stavového vektoru $\overline{x_{t|t}}$:

$$\overline{x_{t|t}} = \overline{x_{t|t-1}} + K_t \cdot \overline{y_t} \tag{4.23}$$

7. rovnice pro zisk kovarianční matici zpětného odhadu $P_{t|t}$:

$$P_{t|t} = \left(I - K_t \cdot H_t^T\right) P_{t|t-1} \tag{4.24}$$

Je vhodné poznamenat, že první dvě rovnice probíhají v rámci prvního stavu, resp. *predikce* a zbylých 5 rovnic probíhá v rámci druhého stavu, resp. *korekce*. Velmi často ještě bývá uvedeno zobecnění prvních dvou rovnic probíhajících v rámci prvního stavu pomocí následující rovnice:

$$x_{t+1} = F \cdot x_t + w_t, \tag{4.25}$$

kde w_t modeluje náhodný proces s charakterem bílého šumu s nulovou střední hodnotou. A dále pak bývá velmi často prováděno zobecnění i procesu probíhajícího v rámci druhého stavu, resp. zbylých pěti rovnic pomocí následující rovnice:

$$z_{t+1} = H \cdot x_t + v_t, \tag{4.26}$$

kde v_t modeluje náhodný proces s charakterem bílého šumu s nulovou střední hodnotou.

Předešlé rovnice obsahují proměnné, jež jsou vázané na systém, pro který je Kalmanův filtr používán, v případě této diplomové práce je tímto systémem sledování obličejů ve videu. Tyto proměnné, resp. tento systém je nutné definovat.

4.10.1 Definice sytému sledování obličejů

Definice systému realizujícího sledování obličejů ve videu probíhá v následujících krocích.

- Definice měřené polohy z_t:

$$z_t = [x, y], \tag{4.27}$$

kde x, resp. y odpovídá x-ové, resp. y-ové souřadnici levého horního rohu obdélníku ohraničujícího sledovaný obličej.

- Definice stavového vektoru x_t:

$$x_t = \lfloor x, y, v_x, v_y, a_x, a_y \rfloor, \tag{4.28}$$

kde v_x, resp. v_y značí rychlost obličeje ve směru osy x, resp. osy y, a_x, resp. a_y značí zrychlení obličeje ve směru osy x, resp. osy y, význam symbolů x a y je identický s jejich významem ve vzorci 4.24.

- Definice matice popisující ideální pohyb obličeje F:

$$F_t = \begin{bmatrix} 1 & 0 & 1 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.29}$$

kdy tato matice vychází z fyzikálních rovnic, jež popisují pohyb objektu:

$$\begin{aligned} x_{t|t} &= x_{t|t-1} + v^x_{t|t-1} \cdot T + \frac{1}{2} a^x_{t|t-1} \\ y_{t|t} &= y_{t|t-1} + v^y_{t|t-1} \cdot T + \frac{1}{2} a^y_{t|t-1} \end{aligned}, \tag{4.30}$$

$$\begin{aligned} v^x_{t|t} &= v^x_{t|t-1} + a^x_{t|t-1} \cdot T \\ v^y_{t|t} &= v^y_{t|t-1} + a^y_{t|t-1} \cdot T \end{aligned}, \tag{4.31}$$

$$\begin{aligned} a^x_{t|t} &= a^x_{t|t-1} \\ a^y_{t|t} &= a^y_{t|t-1} \end{aligned}, \tag{4.32}$$

kde symbol *T* označuje čas mezi jednotlivými vstupními snímky videosekvence, symbol *x*, resp. *y* odpovídá symbolu ze vzorce 4.25, symbol v^x, resp. v^y odpovídá rychlosti sledovaného obličeje ve směru osy *x*, resp. osy *y* a symbol a^x, resp. a^y odpovídá zrychlení sledovaného obličeje ve směru osy *x*, resp. osy *y*.

- Definice převodní matice *H*, jež konkrétně pro systém sledování obličejů ve videu popisuje převod stavového vektoru na polohu sledovaného obličeje. Pro tento konkrétní případ má převodní rovnice následující tvar:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{4.33}$$

- Definice matice *Q*, jež modeluje nepřesnosti vzniklé, resp. šum vzniklý během výpočtů prováděných v průběhu Kalmanova filtru:

$$Q = \left[w_t \cdot w_t^T \right], \tag{4.34}$$

kde význam symbolu w_t je totožný s jeho významem ve vzorci 4.25.

- Definice matice *R*, jež představuje nepřesnosti způsobené, resp. šum zanesený měřením:

$$R = \left[v_t \cdot v_t^T \right] \tag{4.35}$$

kde význam symbolu v_t je totožný s jeho významem ve vzorci 4.26.

4.10.2 Inicializace systému sledování obličejů

Po definici systému sledování tváří ve videu, je nutné ještě tento systém vhodně inicializovat. Inicializace systému je složena z následujících čtyř bodů:

- Inicializace stavového vektoru: V počáteční fázi Kalmanova filtru máme k dispozici pouze informace o souřadnicích obličeje, proto je inicializace stavového vektoru řízena podle následujícího vzorce:

$$\overline{x_0} = \left[x_0 \quad y_0 \quad 0 \quad 0 \quad 0 \quad 0 \right]^T, \tag{4.36}$$

kde $\overline{x_0}$ stavový vektor v čase *t=0*, x_0, resp. y_0 značí *x-ovou*, resp. *y-ovou* souřadnici polohy obličeje v čase *t=0*.

- Inicializace matice *P*: Kovarianční matice odhadu *P* udává míru, na kolik byl odhad stavového vektoru $\overline{x_t}$ nepřesný v čase *t*, kdy větší hodnota udává větší nedůvěru k naměřeným hodnotám. V systému sledování tváří ve videu na začátku běhu systému známe 2 z 6 hodnot

ve stavovém vektoru, z čehož plyne, že 4 hodnoty jsou velmi nedůvěryhodné, z čehož dále plyne, že vhodná inicializační matice *P* by měla být tvořena čtyřmi velmi vysokými čísly [33,34]. Zbylé dvě hodnoty jsou pak rovny hodnotě *r*, kde význam hodnoty *r* prezentuje vzorec 4.40 uvedený v závěru této podkapitoly. Ukázka jedné z vhodných inicializací matice *P* pro systém sledování tváří je dána následujícím vzorcem [34]:

$$P = \begin{bmatrix} r & 0 & 0 & 0 & 0 & 0 \\ 0 & r & 0 & 0 & 0 & 0 \\ 0 & 0 & \varepsilon & 0 & 0 & 0 \\ 0 & 0 & 0 & \varepsilon & 0 & 0 \\ 0 & 0 & 0 & 0 & \varepsilon & 0 \\ 0 & 0 & 0 & 0 & 0 & \varepsilon \end{bmatrix}, \qquad (4.37)$$

kde ε >> *0*, konkrétně $\varepsilon = 10^4$. Použití uvedené matice způsobí, že Kalmanův filtr v úvodním běhu preferuje měřené hodnoty, resp. polohy obličejů před odhadovanými.

- Inicializace matice *Q*: Tato matice udává míru proměnlivosti pohybu sledované tváře od ideální trajektorie, jež je spočtena Kalmanovým filtrem pomocí matice *F*. Větší hodnoty matice *Q* značí větší proměnlivost, což k lepší přizpůsobivosti celého Klamanova filtru. Naproti tomu menší hodnoty matice *Q* znamenají nižší proměnlivost v pohybu sledovaných tváří, což vede k horší přizpůsobivost Kalmanova filtru vůči velkým změnám v pohybu sledovaných tváří. Tyto hodnoty je pak velmi těžké nastavit správně, jelikož záleží na situaci, v jaké bude sledování tváří používané. Vzorec 4.38 prezentuje nastavení, jež bylo zprvu převzato z [33, 34] a následně upraveno při testování této diplomové práce. Konkrétně vzorec 4.38 prezentuje volbu, jež vychází z modelování pohybu objektu s náhodným zrychlením se standardní odchylkou δ_a, viz [33].

$$Q = F \cdot Q_0 \cdot F^T = \begin{bmatrix} \frac{T^4}{4} & 0 & \frac{T^3}{2} & 0 & \frac{T^4}{4} & 0 \\ 0 & \frac{T^4}{4} & 0 & \frac{T^3}{2} & 0 & \frac{T^4}{4} \\ \frac{T^3}{2} & 0 & T^2 & 0 & T & 0 \\ 0 & \frac{T^3}{2} & 0 & T^2 & 0 & T \\ \frac{T^2}{4} & 0 & T & 0 & 1 & 0 \\ 0 & \frac{T^2}{4} & 0 & T & 0 & 1 \end{bmatrix} \cdot \delta_a{}^2, \qquad (4.38)$$

kde *T* označuje čas mezi jednotlivými vstupními snímky, δ_a pro jednoduchost označuje odchylku zrychlení, jak v *x-ovém* směru, tak i v *y-ovém* směru. Je vhodné poznamenat, že zjednodušení δ_a pro oba směry si je možné dovolit, jelikož náhodné zrychlení je náhodné pro

oba směry stejně. Posledním neobjasněným symbolem ze vzorce 4.38 je proměnná Q_0, jež je definována následujícím vzorcem:

$$Q_0 == \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \delta_a & 0 \\ 0 & 0 & 0 & 0 & 0 & \delta_a \end{bmatrix}. \tag{4.38}$$

- Inicializace matice *R*: Matice *R* obecně udává míru nepřesnosti prováděného měření v rámci získávání naměřených hodnot, v systému sledování tváří ve videu je to konkrétně nepřesnost pozic sledovaných obličejů. Tyto obličeje jsou získávány detektorem obličejů ve videu, jenž funguje na principu Viola a Jones. Stejným principem jsou získávány měřená data (pozice obličejů) i ve [34], kde je navíc ověřeno, že tento princip získávání měřených dat detekuje 95% všech obličejů. Pozice těchto detekovaných obličejů se od skutečných obličejů navíc nikdy neliší o více jak 10 pixelů, což nám dává toleranci v rozmezí 10 pixelů. Pokud tyto údaje vyhodnotíme s ohledem na Gaussův šum přidaný v rámci Kalmanova filtru, dostaneme toleranci 6.5 pixelu na jednu pozici takto detekovaného obličeje. Více informací o zisku této hodnoty lze nalézt v [34]. S využitím předešlé hodnoty potom můžeme inicializovat vektor šumu měření v_t, jenž figuroval ve vzorci 4.26 matici dle vzorce 4.39:

$$v_t = \begin{bmatrix} 6.5 \\ 6.5 \end{bmatrix}. \tag{4.39}$$

Dosazením vzorce 4.39 do vzorce 4.26 nám potom vyjde následující vzorec 4.40 pro možnou inicializaci celé matici *R*:

$$R = \left[v_t \cdot v_t^T \right] = \begin{bmatrix} 6.5^2 & 0 \\ 0 & 6.5^2 \end{bmatrix} = \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix}. \tag{4.40}$$

kde *r* odpovídá hodnotě *r*, jež byla použita při výpočtu matice *P* dle vzorce 4.37.

5 Návrh systému detekce a sledování

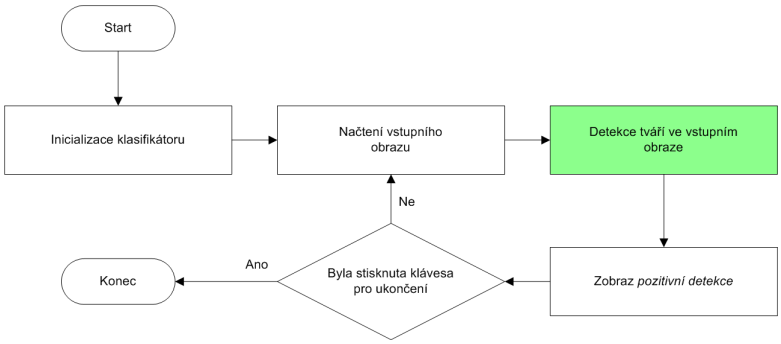
V této kapitole je uveden návrh řešení celé knihovny, jež byla realizována v rámci praktické části diplomové práce. Návrh knihovny je rozdělen do 4 částí, resp. podkapitol. V první podkapitole je popsán návrh řešení detekování obličejů v obraze, ve druhé podkapitole je uveden návrh řešení sledování obličejů v obraze, ve třetí a čtvrté podkapitole jsou uvedeny dva návrhy řešení možného propojení detekce tváří se sledováním tváří. Knihovna realizovaná v rámci praktické části této diplomové práce obsahuje také funkce pro detekci lidské kůže v obraze, přičemž návrh řešení detekce lidské kůže v obraze byl uveden již dříve v této práci, konkrétně ve třetí kapitole orientované výhradně na problematiku detekce lidské kůže v obraze.

Ještě před popisem jednotlivých navržených řešení je vhodné si ujasnit význam slova tvář, kdy slovo tvář bude mít po zbytek páté kapitoly význam obdélníku ohraničujícího lidský obličej. Tento ohraničující obdélník/tvář pak bude tvořen/tvořena čtveřicí čísel mající význam v pořadí, *x-ová* souřadnice levého horního rohu obdélníku/tváře, *y-ová* souřadnice levého horního rohu obdélníku/tváře, šířka obdélníku/tváře, výška obdélníku/tváře.

5.1 Návrh systému detekce tváří

Tato kapitola představuje návrh řešení detekce tváří ve videu. Představený návrh vychází z principu funkce *cvHaarDetectObject* z knihovny OpenCV. Konkrétně z principu funkce *cvHaarDetectObject* používající klasifikátor *haarcascade_frontalface_alt*, proto i navržený detektor tváří ve videu je navržen pro používání klasifikátoru *haarcascade_frontalface_alt*.

Obecná funkce navrženého detektoru je pak rozdělena do čtyř fází, kdy v první fázi je inicializován klasifikátor, ve druhé fázi je načten vstupní obraz ze vstupní videosekvence. Je vhodné poznamenat, že vstupní videosekvence může být mapována jak na vstupní video soubor, tak na připojenou kameru. Ve třetí fázi obecné funkce navrženého detektoru je provedena optimalizovaná detekce tváří ve vstupním obraze, kdy výstupem z této detekce je pole detekovaných tváří. V čtvrté fázi je provedeno zobrazení všech tváří z pole detekovaných tváří vráceného předešlou optimalizovanou detekcí. Poté ve čtvrté fázi následuje vyhodnocení, zda během provádění programu již byla stisknuta klávesa pro ukončení programu. Pokud je zjištěno, že klávesa stisknuta byla, program končí, v opačném případě program pokračuje opět na začátek druhé fáze. Blokové schéma právě popsané obecné funkce je zobrazeno na následujícím obrázku 5.1.



Obrázek 5.1: Blokové schéma detektoru obličejů ve videu.

Kapitola 5.1 je dále rozdělena do 7 podkapitol, kdy v podkapitole 5.1.1 je blíže popsána struktura klasifikátoru *haarcascade_frontalface_alt*. Ze struktury zmíněného klasifikátoru vyplyne skutečnost, že tento klasifikátor je velice rozsáhlý, což přináší velmi vysokou míru úspěšnosti při klasifikování tímto klasifikátorem, na úkor rychlosti této klasifikace. Ze struktury zmíněného klasifikátoru si však nelze vytvořit přesnou představu o způsobu jeho vyhodnocování, proto jsou v této kapitole uvedeny podkapitoly 5.1.2 a 5.1.3, kdy podkapitola 5.1.3 popisuje právě způsob standardní, neoptimalizované, klasifikace klasifikátorem *haarcascade_frontalface_alt*. Klasifikace pomocí zmíněného klasifikátoru však klasifikuje pouze jedno podokno vstupního snímku, proto je vhodné v kapitole 5.1.2 nejprve představit standardní způsob, jakým lze klasifikátor aplikovat při detekci v celém jednom vstupním snímku, a až poté uvést standardní klasifikaci jednoho podokna.

Dále jsou v podkapitolách 5.1.4 a 5.1.5 prezentovány způsoby, jakými je možné standardní řešení detektoru tváří urychlit. V podkapitole 5.1.4 je konkrétně zmíněna možnost urychlení vycházející z optimalizace standardního přístupu k procházení podoken, jenž je použit pro procházení podoken ve standardním řešení detekce tváří v obraze. Podkapitola 5.1.5 pak pro změnu představuje způsob urychlení samotné klasifikace jednoho podokna založený na využití SIMD technologií.

Na konec kapitoly 5.1 jsou poté ještě uvedeny poslední dvě podkapitoly 5.1.6 a 5.1.7, kdy v podkapitole 5.1.6 je uveden princip optimalizovaného detektoru tváří, demonstrující nejvhodnější způsob kombinace všech optimalizací zmíněných v podkapitolách 5.1.4 a 5.1.5. Principy optimalizovaného klasifikování, jež tvoří základ optimalizovaného detektoru tváří, jsou pak popsány v podkapitole 5.1.7.

5.1.1 Struktura použitého klasifikátoru

Tato kapitola popisuje strukturovaný zápis použitého klasifikátoru, tedy strukturovaný zápis souboru *haarcascade_frontalface_alt*, jenž je uložen v xml formátu.

Kořenovým prvkem klasifikátoru je element <*haarcascade*>. Element <*haarcascade*> má pod sebou element <*stages*>. Uvnitř elementu <*stages*> jsou položky označené <_>, jež obsahují informace o jednotlivých stupních kaskády. Z předchozího textu plyne, že jednotlivé stupně kaskády lze symbolizovat zápisem <*stages*><_>, což je při vysvětlování poněkud matoucí, proto o nich

v dalším textu bude psáno jako o elementech typu <*stage*>, přestože takové značení neodpovídá reálnému zápisu v souboru.

Prvek <*stage*> má vnořeny jednotlivé příznaky formou množiny stromů, element <*trees*>. Element <*trees*> pak obsahuje jednotlivé stromy, kdy v každém jednom stromu je uložen jeden Haarův příznak. Přístup k jednotlivým stromům je opět pomocí symbolu <_>. Každý strom, resp. každý prvek <*trees*><_> má jako svůj kořenový prvek element <*feature*>., Z element <*feature*> vychází tři větve, kde středová větev obsahuje prvek <*threshold*>, levá větev obsahuje element <*left_val*> a konečně pravá větev obsahuje prvek <*right_val*>.

Element <feature> udává aplikační oblast Haarova příznaku, pro tyto účely má vnořeny dva elementy, <*rects*> a <*tilted*>. Element <*rects*> obsahuje 2 až 3 položky, kdy každá tato položka je určena pěti čísly. První 4 čísla zastupují levý horní bod obdélníkové oblasti *x-ová* souřadnice, levý horní bod obdélníkové oblasti *y-ová* souřadnice, šířka obdélníkové oblasti, výška obdélníkové oblasti, přičemž všechny tyto hodnoty jsou voleny z rozsahu <0,19> (rozlišení vstupního podokna je 20x20 pixelů). Páté číslo udává váhu obdélníka.

Element <*tilted*> udává úhel natočení, ve stupních, celého příznaku, resp. obou dvou nebo všech tří obdélníků. Za zmínku jistě stojí skutečnost, že v souboru *haarcascade_frontalface_alt* element <*tilted*> nabývá pouze hodnoty 0, z čehož plyne možnost ignorovat tento element.

Elementy <tilted>, <threshold>, <right_val> a <left_val> neobsahují krom své vlastní hodnoty žádné další uzly. Element <*stage*> obsahuje krom elementu <*trees*> ještě elementy <*stage_threshold*>, <*parent*> a <*next*>.

Element <*stage_threshold*> obsahuje pouze jednu, *prahovou*, hodnotu, podle níž se určuje výsledek celého jednoho stupně kaskády. Element <*parent*> obsahuje odkaz na předchozí stupeň kaskády a konečně element <*next*> obsahuje odkaz na následující stupeň kaskády.

Soubor *haarcascade_frontalface_alt* obsahuje 1775 příznaků se dvěma obdélníky, 360 příznaků se třemi obdélníky rozdělených do 22 stupňů kaskády. To nám dává celkem 4630 obdélníků, jež je třeba vyhodnotit pro jednu úspěšnou pozitivní detekci podokna.

Závěrem této kapitoly je vhodné poznamenat, že názorná ukázka jednoho stupně popisovaného klasifikátoru je uvedena v příloze 1 na obrázku P-1.

5.1.2 Standardní detekce tváří v obraze

Tato kapitola popisuje standardní detekci tváří v jednom vstupním obraze, resp. detektor tváří v obraze. Tento detektor během své činnosti používá klasifikátor inicializovaný podle souboru *haarcascade_frontalface_alt*. V souvislosti s předchozí větou je vhodné poznamenat, že inicializace používaného klasifikátoru neprobíhá v rámci detekce tváří. Začlenění zmíněné inicializace do procesu detekce tváří v obraze by totiž způsobilo inicializaci vstupního klasifikátoru po každém načtení vstupního obrazu. Inicializace zmíněného klasifikátoru po každém načtení vstupního obrazu však

není žádoucí, jelikož tato inicializace není na vstupním obraze závislá. Z předchozí věty plyne, že při opakovaném volání detekce tváří by zbytečně probíhal stejný proces vícekrát, což by bylo plýtvání drahocenným strojovým časem. A k tomuto opakovanému detekování tváří skutečně dochází, např. při použití tohoto detektoru v rámci detektoru tváří ve videu. Proto je vhodnější provést tuto zmiňovanou inicializaci mimo proces detekování tváří v obraze.

Popisovaný detektor na svém vstupu očekává jeden vstupní snímek v barevné reprezentaci RGB a jeden korektně inicializovaný klasifikátor. Výstupem tohoto detektoru je pole detekovaných tváři ve vstupním snímku.

Obecný průběh detekce tváří ve vstupním obraze je možné rozdělit do 4 fází. V první fázi je nejprve provedeno načtení požadovaných vstupů, tedy načtení vstupního snímku a vstupního klasifikátoru. Dále je vynulováno pole pozitivních podoken. Poté je proveden převod vstupního snímku na reprezentaci ve stupních šedi, viz kapitola 2.8. V závěru první fáze je ještě provedeno vyhodnocení, zda je načtený klasifikátor korektně inicializovaný. Pokud je vyhodnoceno, že načtený klasifikátor byl inicializovaný korektně, tak popisovaný detektor pokračuje druhou fázi, v opačném případě detektor přejde na začátek 4. fáze.

Ve druhé fázi je nejprve proveden převod šedotónového obrazu na integrální obraz a poté následuje převod integrálního obrazu na obraz druhých mocnin hodnot integrálního obrazu, jenž bude ve zbytku této práce také označován zkráceně jako obraz druhých mocnin. Obraz druhých mocnin má stejné rozměry jako integrální obraz, přičemž výpočet každého jednoho pixelu obrazu druhých mocnin lze provést podle následujícího vzorce:

$$ss(x, y) = ii^2(x, y), \tag{5.1}$$

kde x značí *x-ovou* souřadnici aktuálně počítaného pixelu, y značí *y-ovou* souřadnici aktuálně počítaného pixelu, *ss* je označení obrazu druhých mocnin hodnot integrálního obrazu, *ii* je označení integrálního obrazu, jež reflektuje značení použité ve vzorci 4.1 z kapitoly 4.5.

Ve třetí fázi dochází k postupné klasifikaci všech podoken vstupního šedotónového obrazu. Je vhodné poznamenat, že každé jedno podokno je označeno jako *pozitivní podokno*, resp. *negativní podokno* v případě, že ho standardní klasifikátor jednoho podokna vyhodnotil s výstupem *pozitivní detekce*, resp. *negativní detekce*. Standardní klasifikátor jednoho podokna je popsán v následující podkapitole. Pro každé jedno *pozitivní podokno* je poté určena jeho obdélníková oblast p, jež je tvořena čtveřicí bodů p_x, p_y, p_w, p_h, kde p_x, resp. p_y značí *x-ovou*, resp. *y-ovou* souřadnici levého horního rohu obdélníkové oblasti, p_w, resp. p_h značí šířku, resp. výšku obdélníkové oblasti. Každá tato oblast p je poté dále upravena podle následujícího vzorce:

$$
\begin{aligned}
&\textit{stupeň zmenšení} = \frac{w_{RGB}}{w_{GS}} = \frac{h_{RGB}}{h_{GS}} \\
&p_x = p_x \cdot \textit{stupeň zmenšení} \\
&p_y = p_y \cdot \textit{stupeň zmenšení} \\
&p_w = p_w \cdot \textit{stupeň zmenšení} \\
&p_h = p_h \cdot \textit{stupeň zmenšení}
\end{aligned}
\quad , \qquad (5.2)
$$

kde w_{RGB}, resp. h_{RGB} značí šířku, resp. výšku vstupního RGB obrazu, a w_{GS}, resp. h_{GS} značí aktuální šířku, resp. výšku šedotónového obrazu. Ve třetí fázi je dále každá takto upravená obdélníková oblast *p* přidána do pole pozitivních podoken. V závěru třetí fáze dochází ke zmenšení šedotónového obrazu v daném poměru. Pokud rozměry zmenšeného šedotónového obrazu jsou větší než rozměry podokna, tedy větší než 20x20 pixelů, detektor pokračuje na začátek druhé fáze, v opačném případě pokračuje na začátek čtvrté fáze.

Ve čtvrté fázi dochází ke zpracováním všech pozitivních podoken, jež byly v průběhu detekce uloženy do pole pozitivních podoken. Účelem zmíněného zpracování je především sdružení překrývajících se podoken a odstranění náhodných pozitivních detekcí. Náhodné pozitivní detekce jsou odstraňovány na základě okolních podoken, pokud se v oblasti vyskytuje pouze jeden pozitivní výskyt, je pozitivní detekce odstraněna. Výstupem zmíněného zpracování je pak pole detekovaných tváří.

Obrázek 5.2: Blokové schéma detektoru tváří v obraze

5.1.3 Standardní klasifikace podokna

Jak již bylo několikrát zmíněno, detektor používaný v diplomové práci pro detekování tváří ve videu pracuje s klasifikátorem *haarcascade_frontalface_alt*.

Pro klasifikování podokna (o rozměrech 20x20 pixelů) vstupního obrazu tímto klasifikátorem je nutné na vstup klasifikátoru dodat odpovídající podokno integrálního obrazu, viz kapitola 4.5 a odpovídající podokno obrazu druhých mocnin integrálního obrazu, viz kapitola 5.1.2. Výstupem klasifikování je *ano*, v případě pozitivní detekce ve vstupním podokně, nebo *ne*, v případě negativní detekce ve vstupním podokně.

Klasifikování podokna, resp. vyhodnocení celého klasifikátoru je složeno ze dvou částí. V první části dochází k výpočtu *vnf* (***v**ariance **n**orm **f**actor*) dle následujícího vzorce:

$$\begin{aligned} vnf &= \frac{suma_{ss}}{20 \cdot 20} - \left(\frac{suma_{ii}}{20 \cdot 20} \right)^2 \\ suma_{ii} &= (a_{ii} + d_{ii}) - (b_{ii} + c_{ii}) \\ suma_{ss} &= (a_{ss} + d_{ss}) - (b_{ss} + c_{ss}) \\ a_{ii} &= ii(1,1), a_{ss} = ss(1,1) \\ b_{ii} &= ii(1,18), b_{ss} = ss(1,18) \\ c_{ii} &= ii(18,1), c_{ss} = ss(18,1) \\ d_{ii} &= ii(18,18), d_{ss} = ss(18,18) \end{aligned} \quad , \qquad (5.3)$$

kde význam značek *ii* a *ss* je dán vzorci 4.1 a 5.1. Ve vzorci 5.3 je dále pro korektní význam hodnot 1 a 18 vyžadována adresace v podokně 20x20 v intervalu od 0 do19, kde 1 odpovídá druhé hodnotě zleva, resp. druhé hodnotě shora, a analogicky 18 odpovídá předposlední hodnotě zleva, resp. předposlední hodnotě shora. Hodnota proměnné *vnf* je dále porovnána s 0. Pokud je hodnota proměnné *vnf* menší než nula, je do této proměnné uložena hodnota 1 v opačném případě je proměnná *vnf* dodatečně odmocněna dle následujícího vzorce:

$$vnf = \sqrt{vnf} \; . \qquad (5.4)$$

Ve druhé části dochází k postupnému vyhodnocování jednotlivých stupňů daného klasifikátoru. Klasifikátor se vyznačuje zapojením všech stupňů do kaskády, viz kapitola 4.3. Z kaskádového zapojení plyne, že vrácením negativní detekce libovolným stupněm kaskády okamžitě končí *negativní detekcí* i celá klasifikace podokna. Dále z kaskádového zapojení plyne, že pro *pozitivní detekci* celého podokna je nutné pozitivně klasifikovat vstupní podokno všemi stupni klasifikátoru.

Vyhodnocení jednoho stupně klasifikátoru začíná postupným vyhodnocováním jednotlivých slabých klasifikátoru, z nichž je daný stupeň klasifikátoru složen. Po každém jednom vyhodnocení slabého klasifikátoru je výstupní hodnota tohoto vyhodnocení započtena do součtu vrácených hodnot (přesněji součet všech vrácených hodnot doposud vyhodnocených slabých klasifikátorů). Zmíněný součet bude v dalším textu zastupován proměnnou *SumaS*. Po každém jednom přičtení k proměnné

SumaS následuje porovnání této proměnné s prahem daného stupně klasifikátoru. Je vhodné poznamenat, že tento práh je získán ze souboru *haarcascade_frontalface_alt*, kde každý stupeň kaskády má svůj práh uložen ve vnořeném elementu <*stage_threshold*>. Zmíněný práh daného stupně klasifikátoru bude v dalším textu označován jako proměnná *ThresholdS*. Pokud z předešlého porovnání vyplyne, že hodnota proměnné *SumaS* přesáhla práh *ThresholdS*, tak vyhodnocení daného stupně klasifikátoru končí vrácením *pozitivní detekce*. Z předchozí věty plyne, že při použití klasifikátoru *haarcascade_frontalface_alt* nemusí během vyhodnocování jednoho stupně klasifikátoru dojít k vyhodnocení všech jeho slabých klasifikátorů. Předchozí skutečnost je jednou z výhod použití zmíněného klasifikátoru pro účely klasifikace jednoho podokna, jelikož při použití dalšího OpenCV klasifikátor *haarcascade_frontalface_default* pro stejné účely je nutné vždy pro každé podokno vyhodnotit všechny vnořené slabé klasifikátory. Pokud z porovnání proměnných *SumaS* a *ThresholdS* vyplyne, že hodnota proměnné *SumaS* nepřesáhla práh *ThresholdS*, pak vyhodnocování jednoho stupně klasifikátoru pokračuje vyhodnocením dalšího slabého klasifikátoru v pořadí. Pokud dojde k vyhodnocení všech slabých klasifikátoru a proměnná *SumaS* přesto nepřesáhla práh *ThresholdS*, pak vyhodnocení daného stupně klasifikátoru končí *negativní detekcí*. Vstupním požadavkem pro vyhodnocení jednoho stupně klasifikátoru je pouze podokno integrálního obrazu. Výstupní hodnotou vyhodnocení jednoho stupně klasifikátoru je *ano* v případě pozitivní detekce daným stupněm klasifikátoru, nebo *ne* v případě negativní detekce daným stupněm klasifikátoru.

Vyhodnocení jednoho slabého klasifikátoru začíná vyhodnocením všech obdélníku, ze kterých je daný slabý klasifikátor složen. Poté je provedena suma vrácených hodnot od všech vyhodnocených obdélníků, v dalším textu bude tato suma označována proměnnou *SumaW*. Dále je hodnota proměnné *SumaW* porovnána se součinem proměnné *vnf* a prahu daného slabého klasifikátoru, jenž v dalším textu ponese označení *ThresholdW*. Hodnota proměnné *ThresholdW* je opět získána ze souboru *haarcascade_frontalface_alt*, kde každý slabý klasifikátor má svůj práh uložen ve vnořeném elementu <*threshold*>. Pokud je hodnota proměnné *SumaW* menší než součin $vnf \cdot ThresholdW$, dojde k vrácení příslušné hodnoty *left_val*, v opačném případě dojde k vrácení příslušné hodnoty *right_val*. Hodnoty *left_val* a *right_val* jsou získány ze souboru *haarcascade_frontalface_alt*, kde každý slabý klasifikátor má svou příslušnou hodnotu *left_val*, resp. *right_val* uloženu ve vnořeném elementu <*left_val*>, resp. <*right_val*>. Vstupními požadavky k vyhodnocení jednoho slabého klasifikátoru jsou integrální obraz a hodnota proměnné *vnf*. Výstupní hodnotou vyhodnocení jednoho slabého klasifikátoru je hodnota *left_val* nebo *right_val*, kdy hodnota *let_val* bývá zpravidla nižší v porovnání s hodnotou *right_val* a symbolizuje tak zpravidla nižší pravděpodobnost *pozitivní detekce*.

Vyhodnocení jednoho obdélníku je složeno pouze z výpočtu vážené sumy, dále v textu označované jako *SumaR*. Výpočet hodnoty proměnné *SumaR* se řídí dle následujícího vzorce:

$$
\begin{aligned}
SumaR &= \left[(A+D)-(B+C)\right]\cdot\frac{r_5}{20\cdot 20}\\
A &= ii(r_1, r_2)\\
B &= ii(r_1, r_2 + r_4)\\
C &= ii(r_1 + r_3, r_2)\\
D &= ii(r_1 + r_3, r_2 + r_4)
\end{aligned}
\quad , \qquad (5.5)
$$

kde *ii* značí vstupní integrální obraz, přičemž značení je shodné se značením ve vzorci 4.1 z kapitoly 4.3, r_1, r_2, r_3, r_4 a r_5 symbolizují hodnoty načtené ze souboru *haarcascade_frontalface_alt*, kde každý obdélník má své hodnoty r_1, r_2, r_3, r_4, r_5 uloženy v elementu <*rects*><_>. Podrobnější popis hodnot r_1, r_2, r_3, r_4 a r_5 je uveden v kapitole 5.1.2. Konstanta 20 ve vzorci 5.5 značí šířku, resp. výšku podokna. Vstupním požadavkem pro vyhodnocení jednoho obdélníku je pouze integrální obraz. Výstupní hodnotou vyhodnocení jednoho obdélníku je vážená suma *SumaR*.

Pro lepší pochopení průběhu vyhodnocování celého klasifikátoru je uvedeno následující blokové schéma:

Obrázek 5.3: Blokové schéma standardní klasifikace jednoho podokna.

5.1.4 Návrh optimalizace průchodu podoken

Z metody standardní detekce tváří v obraze představené v kapitole 5.1.2 plyne, že k vyhodnocení vstupního snímku je třeba projít a následně klasifikovat každé podokno tohoto snímku.

Standardní přístup pro procházení podoken demonstrovaný v referenčních příkladech distribuovaných s knihovnou OpenCV lze zapsat následujícím formálním zápisem:

Pro každou kombinaci souřadnic x a y pro niž platí že $x \in N \wedge 0 \leq x < (w - w_{sub}) \wedge y \in N \wedge 0 \leq y < (h - h_{sub})$, kde w odpovídá šířce procházeného obrazu, w_{sub} odpovídá šířce podokna, h odpovídá výšce procházeného obrazu a h_{sub} odpovídá výšce podokna, proveď:

1. Funkcí *cvSetImageROI* nastav oblast zájmu, zkráceně ROI (***r**egion **o**f **i**nterest*), procházeného obrazu na oblast vymezenou obdélníkem začínajícím v bodě $[x,y]$ a končícím v bodě $[x+w_{sub},y+h_{sub}]$.
2. Funkcí *cvCopy* zkopíruj oblast zájmu do podokna.
3. Proveď operaci s podoknem.

Z formálního zápisu je zřejmé, že dochází k neustálému kopírování paměti, což je nevýhodné zejména proto, že kopírování paměti je jednou z nejvíce časově náročných operací vůbec. Dopad této nevýhody roste s rozměry procházených obrazů.

Optimalizovaný přistup pro procházení a následnou klasifikaci podoken se neustálému kopírování vstupního snímku do podoken vyhýbá. Při aplikování optimalizované volby ROI se jako vstup klasifikátoru vždy použije kompletní integrální obraz a kompletní obraz druhých mocnit hodnot integrálního obrazu. Dále se na vstup klasifikátoru předají souřadnice aktuálního podokna, resp. souřadnice aktuální oblasti zájmu ve vstupních snímcích. Je vhodné poznamenat, že při klasifikaci jednoho podokna je oblast zájmu stejná pro oba vstupní obrazy. Tento přístup si v porovnání se standardní klasifikaci žádá změnu způsobu extrahování hodnot a_{ii}, b_{ii}, c_{ii}, d_{ii}, a_{ss}, b_{ss}, c_{ss}, d_{ss} pro účely výpočet proměnné *vnf* a změnu způsobu extrahování hodnot A, B, C, D pro účely vyhodnocování jednoho obdélníku.

Extrahování hodnot a_{ii}, b_{ii}, c_{ii}, d_{ii}, a_{ss}, b_{ss}, c_{ss}, d_{ss} z kompletních vstupních snímků pro účel výpočtu proměnné *vnf* při aplikování optimalizované volby ROI probíhá podle vzorce 5.6, místo standardního způsobu extrahování těchto hodnot, jež je použit ve vzorci 5.3 (v podkapitole 5.1.3):

$$\begin{aligned} a_{ii} &= ii(1 + x_{ROI}, 1 + y_{ROI}), a_{ss} = ss(1 + x_{ROI}, 1 + y_{ROI}) \\ b_{ii} &= ii(1 + x_{ROI}, 18 + y_{ROI}), b_{ss} = ss(1 + x_{ROI}, 18 + y_{ROI}) \\ c_{ii} &= ii(18 + x_{ROI}, 1 + y_{ROI}), c_{ss} = ss(18 + x_{ROI}, 1 + y_{ROI}) \\ d_{ii} &= ii(18 + x_{ROI}, 18 + y_{ROI}), d_{ss} = ss(18 + x_{ROI}, 18 + y_{ROI}) \end{aligned}, \qquad (5.6)$$

kde x_{ROI}, resp. y_{ROI} odpovídá *x-ové*, resp. *y-ové* souřadnici levého horního rohu oblasti zájmu, význam ostatních symbolů se přesně shoduje s jejich významem ve vzorci 5.3.

Extrahování hodnot *A, B, C, D* pro účely vyhodnocování jednoho obdélníka pak probíhá podle vzorce 5.7, místo standardního způsobu extrahování těchto hodnot, jež byl použit ve vzorci 5.5.

$$\begin{aligned} A &= ii(t_1, t_2) \\ B &= ii(t_1, r_2 + t_4) \\ C &= ii(r_1 + t_3, t_2) \\ D &= ii(r_1 + t_3, r_2 + t_4) \\ t_1 &= r_1 + x_{ROI}, t_2 = r_2 + y_{ROI} \\ t_3 &= r_3 + x_{ROI}, t_4 = r_4 + y_{ROI} \end{aligned} \quad , \qquad (5.7)$$

kde x_{ROI}, resp. y_{ROI} odpovídá *x-ové*, resp. *y-ové* souřadnici levého horního rohu aktuálního podokna, význam ostatních symbolů se přesně shoduje s jejich významem ve vzorci 5.5.

Časová náročnost výpočtu 4 součtů je nižší v porovnání s kopírováním 20x20 pixelů, což dokládá i skutečnost cca 34% urychlení detekce tváří při použití optimalizovaného průchodu podoken oproti standardní detekci pro obrazy v rozlišení 800x600 pixelů s průměrným počtem tváří. Podrobnější srovnání obou zmíněných přístupů k procházení podoken je uvedeno v kapitole 7, konkrétně v tabulce 7.2. Kapitola 5.1.5.1 dále prezentuje možnost, jak lze čtyři nově přidané operace sčítání ještě dále optimalizovat s využitím SIMD instrukcí.

Na závěr této kapitoly je ještě vhodné zmínit, že je možný také jiný přístup ke klasifikování vstupního obrazu, a to přístup, kdy není procházeno a následně klasifikováno úplně každé podokno vstupního obrazu. Tímto přístupem je možné klasifikovat například každé druhé, resp. třetí, resp. *n-té* podokno v *x-ovém*, resp. *y-ovém* směru. Je zřejmé, že zmíněným přístupem je snížena přesnost detektoru za cenu navýšení rychlosti detekce. Klasifikaci každého druhého, resp. třetího, resp. *n-tého* podokna v *x-ovém* směru je obecně známá pod pojmem klasifikace s krokem 2, resp. 3, resp. *n* v *x-ovém* směru. Analogicky pak klasifikace každého druhého, resp. třetího, resp. *n-tého* podokna v *y-ovém* směru je obecně známá pod pojmem klasifikace, popřípadě procházení podoken, s krokem 2, resp. 3, resp. *n* v *y-ovém* směru. Pokud u kroku klasifikace, resp. procházení podoken ve vstupním obraze není uvedena specifikace *x-ového* ani *y-onového* směru, pak je bráno, že se jedná o klasifikaci, resp. procházení podoken, jež má v obou směrech velikost kroku stejnou. Nyní je možné upřesnit, že OpenCV standardní přístup pro procházení podoken používá krok o velikosti 2. Optimalizovaný přístup pro procházení podoken proto bude rovněž používat krok 2. Vliv velikosti kroku procházení podoken na rychlost a přesnost detektoru tváří v obraze je graficky shrnut v kapitole 7.2.1.

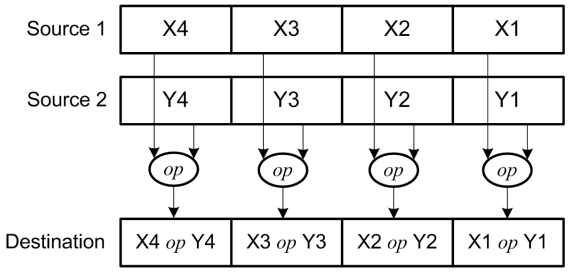
5.1.5 Návrh optimalizace pomocí SIMD

Na začátek této kapitoly je vhodné poznamenat, že podrobný popis SIMD (***s**imple **i**nstruction **m**ultiple **d**ata*) instrukcí není cílem této práce, jelikož problematika SIMD technologií je natolik rozsáhlé téma, že ho není možné v této práci kompletně probrat, podrobné informace je však možné

nalézt např. v [31]. Snahou této kapitoly je pouze ukázka základních principů SIMD instrukci, a dále prezentace možných optimalizací vyhodnocování klasifikátoru pomocí těchto instrukcí.

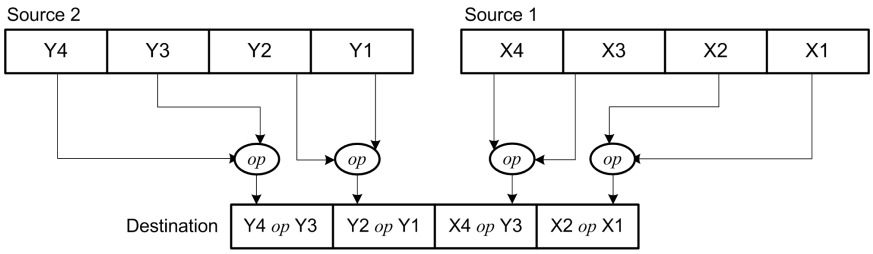
Základní CPU instrukce provádějí jednu operaci během jedné CPU instrukci, oproti tomu princip SIMD instrukcí spočívá v souběžném provádění více operací během jedné SIMD instrukce. Tyto instrukce pracují se speciálními SIMD registry, jež vyžadují speciální uspořádání zpracovávaných dat. SIMD registry dělí data do logických bloků, kdy počet těchto bloků odpovídá počtu souběžně prováděných operací nad těmito registry. SIMD registry mohou při SIMD instrukcích fungovat jako zdrojové nebo cílové, počet zdrojových a cílových registrů je závislý na druhu SIMD instrukcí. Do logických bloků SIMD registru mohou být ukládány celá i reálná čísla s různou přesností, druh uložených čísel se ale musí shodovat ve všech blocích celého registru. SIMD se pak také odlišují podle druhu dat, s nimiž pracují, více v [31]. SIMD instrukce se dále odlišují druhem operací, jež provádějí a také směrem, ve kterém tyto operace provádějí. Příkladem rozlišení SIMD instrukcí na základě druhu prováděné operace může být např. odlišení SIMD instrukcí provádějící paralelně *n* operací *součet* nad *n* bloky dat příslušných registru od SIMD instrukcí provádějící paralelně *n* operací *rozdíl* nad *n* bloky dat příslušných registru, atd. Dělení SIMD instrukcí dle směru, ve kterém souběžně provádějí operace je binární, existují pouze vertikální SIMD instrukce a horizontální SIMD instrukce. Vertikální SIMD instrukce tvoří naprostou většinu všech SIMD instrukcí, proto pokud nebude explicitně uvedeno jinak, bude SIMD instrukce automaticky považována za vertikální SIMD instrukci. Vertikální SIMD instrukce, jak již jejich název říká, jsou prováděny odshora dolu a pro svůj běh potřebují minimálně jeden zdrojový a jeden cílový prostor. Horizontální SIMD instrukce jsou oproti tomu prováděny zleva doprava a nejčastěji v rámci pouze jednoho SIMD registru, jenž pak funguje jako zdrojový i cílový SIMD registr současně.

Obrázek 5.4, prezentuje funkci jedné **vertikální** SIMD instrukce realizující paralelní provedení čtyř operací *op* nad třemi SIMD registry, dvěma zdrojovými *Source 1* a *Source 2* a jedním cílovým *Destination*. Všechny tři registry jsou rozděleny do čtyř logických bloků.



Obrázek 5.4: Model vertikální SIMD operace nad dvěma MMX, resp. SSE registry [31].

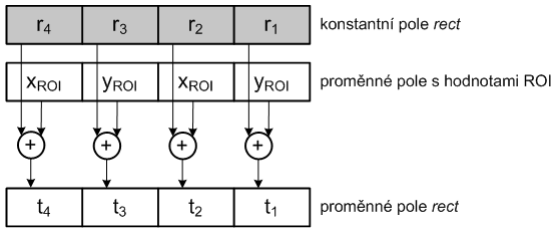
Obrázek 5.5, prezentuje funkci jedné **horizontální** SIMD instrukce realizující paralelní provedení čtyř operací *op* nad třemi SIMD registry, dvěma zdrojovými *Source 1* a *Source 2* a jedním cílovým *Destination*. Všechny tři registry jsou rozděleny do čtyř logických bloků.



Obrázek 5.5: Model horizontální SIMD operace nad dvěma MMX, resp. SSE registry.

5.1.5.1 SIMD na úrovni obdélníku

Tato kapitola prezentuje možnost aplikování SIMD instrukcí pro výpočet proměnné *SumaR*, konkrétně pro výpočet proměnných t_1, t_2, t_3, t_4, jež figurují ve vzorci 5.7 pro výpočet proměnné *SumaR*. Zmíněný vzorec pro výpočet proměnné *SumaR* pomocí hodnot t_1, t_2, t_3, t_4 je uveden v kapitole 5.1.4, proto zde nebude již znovu opakován. Princip aplikováním SIMD instrukcí na výpočet proměnných t_1, t_2, t_3, t_4 demonstruje následující obrázek, přičemž značení v obrázku se shoduje se značením použitým v kapitolách 5.1.3 a 5.1.4.



Obrázek 5.6: SIMD výpočet hodnot t_1, t_2, t_3, t_4.

Pro nejvyšší možnou efektivitu tohoto způsobu aplikace SIMD instrukcí je vhodné během fáze inicializace načítat hodnoty r_1, r_2, r_3, r_4 do předem připravených zarovnaných polí v paměti. Poté je možné provést SIMD instrukci bez vedlejší režie spojené se zarovnáním paměti před plněním SIMD registrů. Přesto však tento přístup nepřináší žádné urychlení, jelikož provedení pouze jedné SIMD instrukce nedokáže vykoupit nároky spojené s plněním SIMD registrů, proto je vhodné nalézt jiný prostor, ve kterém by bylo možné nad jednou naplněným SIMD registrem provést více SIMD operací po sobě. Nalezený prostor pro efektivnější uplatnění SIMD instrukcí popisují následující dvě kapitoly. Před koncem této části je vhodné ještě zmínit, že tato možnost využívání SIMD instrukcí byla do realizovaného detektoru implementována zejména pro srovnávací účely, jelikož srovnávací tabulky mají poté větší vypovídající hodnotu, resp. čtenář si tak může vytvořit lepší představu o hranici, kdy se již používání SIMD vyplatí, a kdy ještě ne. Zmíněné výsledky jsou presentovány v tabulce 7.2 v kapitole 7.2.

5.1.5.2 SIMD na úrovni slabého klasifikátoru

Tato kapitola představuje možnost aplikování SIMD instrukcí pro vyhodnocení jednoho slabého klasifikátoru se třemi vnořenými obdélníky. Standardní způsob vyhodnocování jednoho slabého

klasifikátoru s libovolným počtem vnořených obdélníků je popsán v kapitole 5.1.3. Optimalizace slabých klasifikátorů se dvěma vnořenými obdélníky je prováděna v rámci celého jednoho stupně klasifikátoru. Optimalizace jednoho stupně klasifikátoru bude popsána v následující kapitole.

SIMD je zde aplikováno pro současné vyhodnocování všech tří obdélníků najednou. Princip této optimalizace je znázorněn na obrázku 5.5, v němž jsou jednotlivé vnořené obdélníky zvýrazněny barvami, kdy zelenou, resp. bílou, resp. žlutou barvou jsou podbarveny bloky zpracovávající první, resp. druhý, resp. třetí obdélník vyhodnocovaného slabého klasifikátoru. Šedě označené bloky v obrázku 5.7 zůstávají nevyužité, resp. jsou plněny samými nulami.

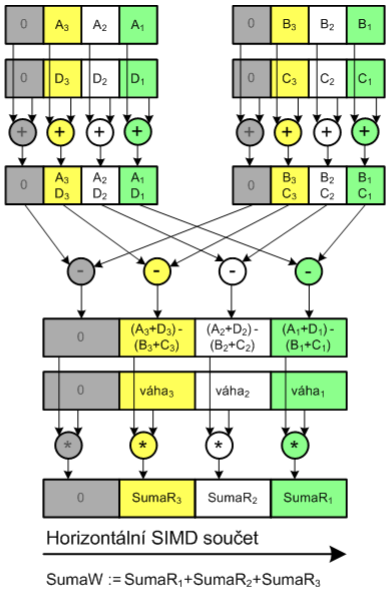
Pro efektivní použití této optimalizace je vyžadováno nejprve korektně připravit paměť pro rychlé plnění 5 SIMD registrů v průběhu vyhodnocování. Těchto 5 SIMD registrů je používáno pro seskupení 5x čtyřech hodnot *A*, *B*, *C*, *D*, r_5, kde první registr shlukuje 4 hodnoty *A*, druhý 4 hodnoty *B*, třetí 4 hodnoty *C*, čtvrtý 4 hodnoty *D* a konečně pátý shlukuje 4 hodnoty r_5. První, resp. druhá, resp. třetí čtveřice hodnot *A*, *B*, *C*, *D* je extrahována ze vstupního podokna integrálního obrazu podle prvního, resp. druhého, resp. třetího obdélníku vnořeného do právě zpracovávaného slabého klasifikátoru. Extrahování jedné čtveřice hodnot *A*, *B*, *C*, *D* dle jednoho obdélníka využívá výše zmíněnou optimalizaci volby ROI, což znamená, že extrahování jedné čtveřice hodnot *A*, *B*, *C*, *D* probíhá dle vzorce 5.7 uvedeným v kapitole 5.1.4. Jako čtvrtá čtveřice hodnot *A*, *B*, *C*, *D*, je použita čtveřice 0. První, resp. druhá, resp. třetí hodnota r_5 poté odpovídá páté hodnotě prvního, resp. druhého, resp. třetího obdélníka vnořeného do právě zpracovávaného slabého klasifikátoru. Značení a význam hodnoty r_5 odpovídá značení a významu, použitém v podkapitole 5.1.1 a v podkapitole 5.1.3. Čtvrtá hodnota r_5 je vždy rovna 0.

Optimalizované vyhodnocení slabého klasifikátoru se třemi obdélníky probíhá ve 3 fázích. V první fázi dochází ke kopírování hodnot z korektně připravené zarovnané paměti do registrů 1 až 5.

Ve druhé fázi je postupně proveden SIMD součet registrů 1 a 4, výsledek součtu je uložen do dočasného registru *regAD*, poté SIMD součet registrů 2 a 3, výsledek součtu je uložen do dočasného registru *regBC*, dále SIMD rozdíl registrů *regAD* a *regBC*, výsledek rozdílu je uložen do dočasného registru *regABCD*. Závěrem je provedena operace SIMD násobení s registrem *regABCD* a registrem 5. Výsledek poslední operace násobení je uložen do dočasného registru *regSumaR*. První, resp. druhý resp. třetí blok registru *regSumaR* obsahuje hodnotu *SumaR* odpovídající výstupní hodnotě prvního, resp. druhého, resp. třetího obdélníku vnořeného do aktuálně zpracovávaného slabého klasifikátoru. Postup neoptimalizovaného výpočtu výstupní hodnoty libovolného obdélníku je popsán v kapitole 5.1.3. Čtvrtý blok registru *regSumaR* obsahuje samé 0.

Ve třetí fázi je horizontální SIMD instrukce pro horizontální součet provedena suma všech čtyř bloků registru *regSumaR*. Výsledek je pak vrácen do výstupního registru a odpovídá hodnotě *SumaW* z kapitoly 5.1.3. Tímto je dokončena optimalizovaná část výpočtu jednoho slabého klasifikátoru se třemi vnořenými obdélníky. Dále už jen následuje standardní výpočet hodnoty *WeakVal* z hodnoty *SumaW*, jenž byl popsán v kapitole standardní klasifikace.

Tato optimalizace slouží pouze pro optimalizovaný výpočet slabých klasifikátorů se třemi obdélníky, proto je vhodné jí zkombinovat s další optimalizací, jež bude představena v další kapitole.



*Pozn.: SumaW značí sumu všech obdélníků slabého klasifikátoru, jelikož se jedná o vyhodnocení slabého klasifikátoru typu 3, je tato suma složena ze součtu třech obdélníku (zeleného, bílého a žlutého)

Obrázek 5.7: SIMD výpočet hodnoty *SumaW* slabého klasifikátoru typu 3.

5.1.5.3 SIMD na úrovni stupně klasifikátoru

Tato kapitola představuje možnost aplikování SIMD instrukcí pro vyhodnocení jednoho stupně klasifikátoru, kdy standardní způsob vyhodnocování jednoho stupně klasifikátoru byl popsán v kapitole 5.1.3. Tato optimalizace spočívá v souběžném vyhodnocení až čtyř slabých klasifikátorů se dvěma obdélníky pomocí SIMD technologie. Dále tato kapitola zavádí pojem slabý klasifikátor typu 2, resp. slabý klasifikátor typu 3, jenž označuje slabý klasifikátor se dvěma vnořenými obdélníky, resp. slabý klasifikátor se třemi vnořenými klasifikátory. Zavedení těchto pojmů složí pro snazší orientaci v textu a význam těchto pojmů zůstane platný po zbytek této práce, bez ohledu na kapitolu, v níž budou tyto pojmy použity.

Metoda současného vyhodnocování 4 slabých klasifikátorů se dvěma vnořenými obdélníky vyžaduje nejprve korektně připravit paměť pro rychlé plnění 14 SIMD registrů v průběhu vyhodnocování. Účel těchto 14 SIMD registrů blíže objasňuje následující odstavec.

Prvních pět registrů je používáno pro seskupení 5x čtyřech hodnot *A*, *B*, *C*, *D*, r_5, kde první registr shlukuje 4 hodnoty *A*, druhý 4 hodnoty *B*, třetí 4 hodnoty *C*, čtvrtý 4 hodnoty *D* a konečně pátý shlukuje 4 hodnoty r_5. Každá čtveřice hodnot *A*, *B*, *C*, *D* je extrahována ze vstupního podokna integrálního obrazu podle **prvního** obdélníka každého ze 4 současně zpracovávaných slabých klasifikátorů typu 2. Extrahování jedné čtveřice hodnot *A*, *B*, *C*, *D* dle jednoho obdélníka využívá výše zmíněnou optimalizaci volby ROI, což znamená, že extrahování jedné čtveřice hodnot *A*, *B*, *C*, *D* probíhá dle vzorce 5.7 uvedeného v kapitole 5.1.4. Každá ze čtyř hodnot r_5 poté odpovídá páté

hodnotě prvního obdélníka každého ze 4 paralelně zpracovávaných slabých klasifikátorů typu 2. Značení a význam hodnoty r_5 odpovídá značení a významu, použitém v kapitole 5.1.3.

Dalších 5 registrů slouží analogicky jako prvních 5, s tím rozdílem, že ve vztazích pro získávání hodnot *A*, *B*, *C*, *D*, r_5 figurují čtyři **druhé** obdélníky čtyř současně zpracovávaných slabých klasifikátorů typu 2, místo čtyř prvních obdélníků stejných klasifikátorů.

Další registr, s pořadovým číslem 11, slouží pro seskupení čtyř hodnot *left_val* získaných ze čtyř zpracovávaných slabých klasifikátorů typu 2. Registry 12 a 13 jsou plněny analogicky jako registr 11, s tím rozdílem, že 12. registr skladuje 4 hodnoty *right_val* a 13. registr ukládá 4 hodnoty *práh*. Způsob získání hodnot *left_val*, *right_val* a *práh* pro každý jeden slabý klasifikátor typu 2 je opět popsáno v kapitole 5.1.1.

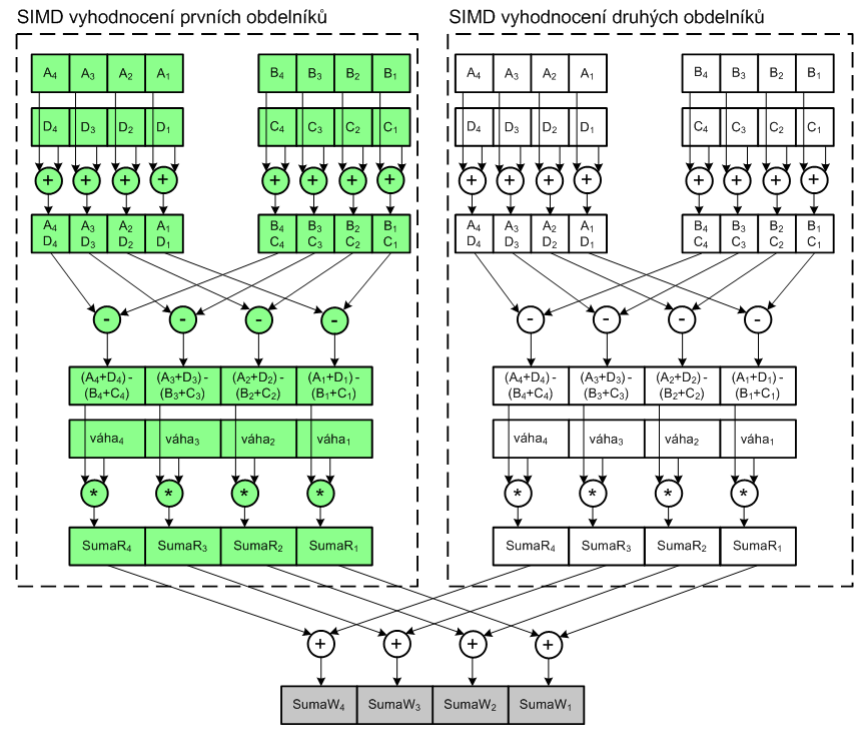
Poslední úložný registr, číslo 14, je plněn 4 hodnotami proměnné *vnf*, kdy výpočet proměnné *vnf* probíhá pouze jednou pro každé podokno, z čehož plyne, že hodnota proměnné *vnf* je pro všechny slabé klasifikátory typu 2 i 3 stejná. Tento registr tedy postačí naplnit pouze jednou před vyhodnocením celého podokna, zatímco všechny předešlé registry je nutné plnit opakovaně před vyhodnocením každé jedné sady 4 slabých klasifikátorů typu 2.

Ještě před samotným rozborem vyhodnocování je vhodné poznamenat, že 14 registrů popsaných v předešlém odstavci nemusí být v průběhu vyhodnocování pořád aktuálních, jelikož jeden fyzicky SIMD registr je ve skutečnosti v jedné fázi vyhodnocování používán např. jako úložiště 4 hodnot *A*, zatímco ve druhé fázi vyhodnocování pak funguje např. jako úložiště 4 hodnot *left_val*. Pro lepší orientaci v registrech je však tato kapitola vysvětlována s iluzí, že účel každého popisovaného SIMD registru zůstává neměnný během celého průběhu paralelního vyhodnocování jedné sady 4 slabých klasifikátorů se dvěma vnořenými obdélníky. Zmíněná iluze umožní možnost odkazovat se v textu na tyto registry jejich pořadovým číslem, tedy pokud bude v pozdějším textu psáno např. o registru číslo 11, bude míněn registr shromažďující čtyři hodnoty *left_val*, jež patří ke čtyřem aktuálně zpracovávaným slabým klasifikátorům typu 2.

SIMD vyhodnocování jedné sady 4 slabých klasifikátorů typu 2 je složeno z 5 fází. V první fázi dochází ke kopírování hodnot z korektně připravené zarovnané paměti do registru 1-5. Poté je postupně proveden SIMD součet registrů 1 a 4, výsledek součtu je uložen do dočasného registru $regAD_1$, poté SIMD součet registrů 2 a 3, výsledek součtu je uložen do dočasného registru $regBC_1$, dále SIMD rozdíl registrů $regAD_1$ a $regBC_1$, výsledek rozdílu je uložen do dočasného registru $regABCD_1$, a závěrem je provedena operace SIMD násobení s registrem $regABCD_1$ a registrem 5. Výsledek poslední operace násobení je uložen do dočasného registru $regSumaR_1$. Registr $regSumaR_1$ je tvořen čtyřmi hodnotami *SumaR*. První hodnota *SumaR* odpovídá výstupní hodnotě prvního obdélníku, vnořeného do prvního zpracovávaného slabého klasifikátoru typu 2. Druhá, resp. třetí, resp. čtvrtá hodnota *SumaR* pak analogicky odpovídá výstupní hodnotě prvního obdélníku vnořeného do druhého, resp. třetího, resp. čtvrtého zpracovávaného slabého klasifikátoru typu 2. Postup standardního výpočtu výstupní hodnoty libovolného obdélníka je popsán v kapitole 5.1.3.

Ve druhé fázi dochází ke kopírování hodnot z korektně připravené zarovnané paměti do registrů 6 až 10. Poté je postupně proveden SIMD součet registrů 6 a 9, výsledek součtu je uložen do dočasného registru *regAD*$_2$, SIMD součet registrů 7 a 8, výsledek součtu je uložen do dočasného registru *regBC*$_2$, SIMD rozdíl registrů *regAD*$_2$ a *regBC*$_2$, výsledek rozdílu je uložen do dočasného registru *regABCD*$_2$ a SIMD násobení registru *regABCD*$_2$ s registrem 10. Výsledek poslední operace násobení je uložen do dočasného registru *regSumaR*$_2$. Registr *regSumaR*$_1$ je tvořen čtyřmi hodnotami *SumaR*. První hodnota *SumaR* odpovídá výstupní hodnotě druhého obdélníku vnořeného do prvního zpracovávaného slabého klasifikátoru typu 2. Druhá, resp. třetí, resp. čtvrtá hodnota *SumaR* pak analogicky odpovídá výstupní hodnotě druhého obdélníku vnořeného do druhého, resp. třetího, resp. čtvrtého zpracovávaného slabého klasifikátoru typu 2.

Ve třetí fázi je proveden SIMD součet registrů *regSumaR*$_1$ a *regSumaR*$_2$ a výsledek je uložen do registru *regSumaW*. Registr *regSumaW* je tvořen čtyřmi hodnotami *SumaW*, kde každá jedna hodnota *SumaW* náleží ke každému jednomu zpracovávanému slabému klasifikátoru typu 2. Postup neoptimalizovaného výpočtu jedné hodnoty *SumaW* je uveden v kapitole 5.1.3.

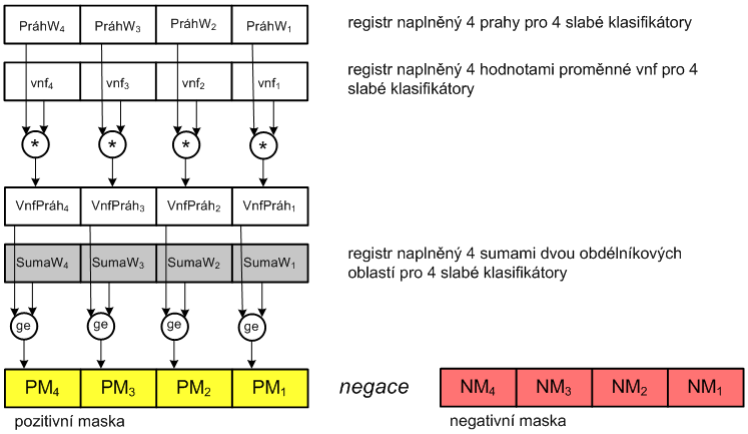


*Pozn.: SumaW značí sumu všech obdélníků slabého klasifikátoru, jelikož se jedná o vyhodnocení slabého klasifikátoru typu 2, je tato suma složena pouze ze součtu dvou obdélníku

Obrázek 5.8: SIMD výpočet čtyř hodnot *SumaW* čtyř slabých klasifikátorů typu 2.

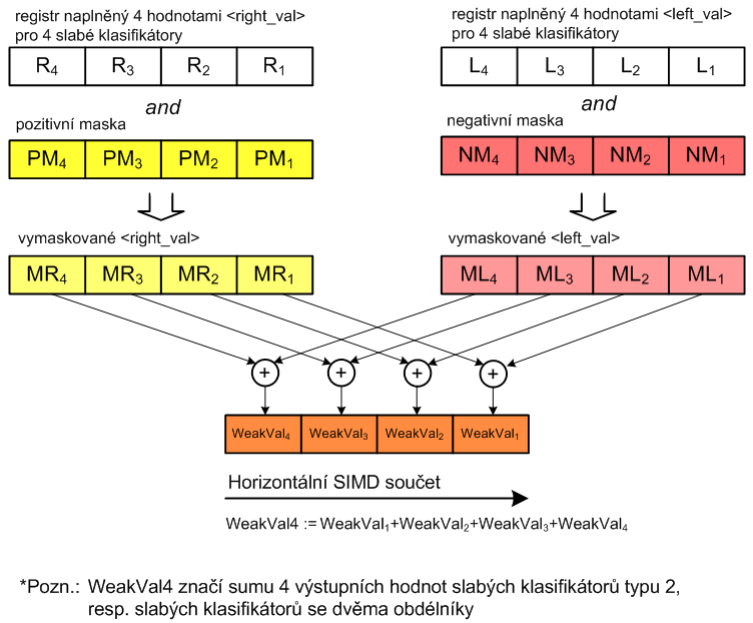
Ve čtvrté fázi dochází ke kopírování hodnot z korektně připravené zarovnané paměti do registrů 13 a 14 a poté jsou tyto registry spolu vynásobeny pomocí SIMD operace násobení. Výsledek předešlého násobení je uložen do registru *regVnfPráh*. Registr *regVnfPráh* je tvořen čtyřmi hodnotami *VnfPráh*. Registr *regSumaW* je poté porovnán s registrem *regVnfPráh* pomocí SIMD

operace větší rovno a výsledek tohoto porovnání je uložen do registru *regPozitivníMaska*. Je vhodné poznamenat, že předešlá operaci způsobí naplnění prvního bloku registru *regPozitivníMaska* samými jedničkami v případě, že první hodnota *SumaW* byla větší rovna první hodnotě *VnfPráh*, v opačném případě bude první blok vyplněn samými nulami. Analogicky budou naplněny i bloky 2, 3 a4 registru *regPozitivníMaska*. Závěrem čtvrté fáze je ještě provedena negace registru *regPozitivníMaska*, jejíž výsledek je uložen do registru *regNegativníMaska*.



Obrázek 5.9: SIMD výpočet *pozitivní* a *negativní* masky.

V páté fázi dochází ke kopírování hodnot z korektně připravené zarovnané paměti do registrů 11 a 12. Poté je provedena SIMD operace logický *and* mezi registry 12 a *regPozitivníMaska*. Výsledek předešlé operace je uložen do registru *regMaskovanéRightVal*. Dále v páté fázi je provedena znovu SIMD operace logický *and*, tentokrát však mezi registry 11 a *regNegativníMaska*. Výsledek předešlé operace je uložen do registru *regMaskovanéLeftVal*. Pro lepší chápání je vhodné poznamenat, že v aktuálním stádiu páté fáze obsahuje registr *regMaskovanéRightVal* hodnoty *right_val* v blocích jež má registr *regMaskovanéLeftVal* vyplněny samými nulami a naopak v blocích, v nichž má registr *regMaskovanéLeftVal* hodnoty *left_val* obsahuje registr *regMaskovanéRightVal* pouze samé nuly. Dalším krokem páté fáze je SIMD součet registrů *regMaskovanéLeftVal* a *regMaskovanéRightVal*. Výsledek předešlého sčítání je uložen do registru *regWeakVal*. Registr *regWeakVal* je tvořen čtyřmi hodnotami *WeakVal*. První hodnota *WeakVal* odpovídá výstupní hodnotě prvního zpracovávaného slabého klasifikátoru typu 2. Druhá, resp. třetí, resp. čtvrtá hodnota *WeakVal* pak analogicky odpovídá výstupní hodnotě druhého, resp. třetího, resp. čtvrtého zpracovávaného slabého klasifikátoru typu 2. Postup neoptimalizovaného výpočtu jedné hodnoty *WeakVal* je uveden v kapitole 5.1.3. Posledním krokem páté fáze je horizontální SIMD instrukce provádějící horizontální součet všech bloků registru *regWeakVal*. Výsledek předchozí operace je vrácen do výstupního registru a odpovídá sumě výstupních hodnot čtyř právě zpracovaných slabých klasifikátorů se dvěma vnořenými obdélníky. Tato suma v dalším textu ponese označení *WeakVal4*.



*Pozn.: WeakVal4 značí sumu 4 výstupních hodnot slabých klasifikátorů typu 2, resp. slabých klasifikátorů se dvěma obdélníky

Obrázek 5.10: SIMD výpočet hodnoty *WeakVal4*.

Nevýhoda právě popsané optimalizace pramení z její omezenosti pouze na slabé klasifikátory se dvěma vnořenými obdélníky, což nám však nevadí, jelikož optimalizace pro vyhodnocování zbylých slabých klasifikátorů, tedy slabých klasifikátorů typu 3, je známá z předchozí kapitoly. O způsobu jak vhodně zkombinovat optimalizované vyhodnocování slabých klasifikátorů typu 2 s optimalizovaným vyhodnocováním slabých klasifikátorů typu 3 je psáno v následujícím odstavci.

5.1.6 Optimalizovaná detekce tváří v obraze

Optimalizovaná detekce tváří v obraze demonstruje nejvhodnější způsob optimalizace standardní detekce tváří v obraze známé z kapitoly 5.1.2. Optimalizovaný detektor tváří v obraze kombinuje optimalizace představené předešlými kapitolami, kdy konkrétně využívá optimalizované procházení podoken a dále optimalizovanou klasifikaci těchto podoken. Je vhodné poznamenat, že optimalizovaná klasifikace podokna bude důkladně popsána v následující kapitole.

Optimalizovaný detektor tváří v obraze pro svou činnost používá stejný klasifikátor jako standardní detektor tváři v obraze. Inicializace klasifikátoru není prováděna v rámci optimalizovaného detektoru, stejně tomu tak je i u standardního detektoru. Vstupní požadavky optimalizovaného detektoru jsou rovněž stejné jako vstupní požadavky standardního detektoru. Od optimalizovaného detektoru můžeme také očekávat stejný výstup jako od standardního detektoru.

Obecný průběh optimalizované detekce tváří ve vstupním obraze je možné rozdělit do 4 fází. První dvě fáze se plně shodují s prvními dvěma fázemi standardní detekce. Třetí fáze optimalizovaného klasifikátoru se od třetí fáze standardní detekce liší pouze ve způsobu procházení podoken vstupních obrazů, kdy standardní obecná funkce používá standardní procházení popsané

v kapitole 5.1.4, zatímco optimalizovaná funkce používá optimalizované procházení podoken, jež bylo popsáno rovněž v kapitole 5.1.4.

Čtvrté fáze obou detektorů se navzájem liší pouze ve způsobu klasifikování podoken, kdy standardní obecná funkce používá standardní klasifikování podoken popsané v kapitole 5.1.3, zatímco optimalizovaná funkce používá optimalizované klasifikování podoken popsané v následující kapitole.

Závěrem této kapitoly je vhodné poznamenat, že detektor realizující optimalizovanou obecnou funkci detekuje obličeje ve vstupních snímcích o cca 50% rychleji než detektor realizující standardní obecnou funkci, přičemž oba detektory dosahují naprosto stejné míry úspěšnosti detekování. Více podrobností k porovnání těchto dvou detektorů je uvedeno v kapitole 7.

5.1.7 Optimalizovaná klasifikace podokna

Tato kapitola popisuje optimalizovanou klasifikaci jednoho podokna. Optimalizovaná klasifikace jednoho podokna je v této práci uvedena mimo jiné jako demonstrace nejvhodnějšího způsobu propojení SIMD optimalizací. V metodě optimalizovaného klasifikování je konkrétně použita optimalizace SIMD na úrovni stupně klasifikátoru a na úrovni slabého klasifikátoru, z čehož plyne, že v optimalizované obecné funkci není použita optimalizace SIMD na úrovni obdélníka. Neaplikování optimalizace SIMD na úrovni obdélníka je úmyslné, jelikož aplikace SIMD na úrovni obdélníka měla na optimalizovaný klasifikátor jednoho podokna spíše negativní dopad. Je vhodné poznamenat, že pro lepší pochopení textu, je v závěru této podkapitoly na obrázku 5.11 uvedeno blokového schéma optimalizovaného klasifikátoru.

Optimalizovaná klasifikace používá stejný klasifikátor, jaký je používán při standardní klasifikaci, viz kapitola 5.1.3, proto od ní můžeme očekávat stejné výstupy jako ze standardní metody klasifikace. Optimalizovaná klasifikace však má jiné požadavky na vstup, kdy místo podokna integrálního obrazu a podokna druhých mocnin hodnot integrálního obrazu, očekává kompletní integrální obraz, kompletní obraz druhých mocnit hodnot integrálního obrazu a souřadnice aktuálního podokna, resp. souřadnice aktuální oblasti zájmu (ROI).

Samotná činnost optimalizovaného klasifikování začíná korektním načtením všech vstupů, dále pokračuje výpočtem proměnné *vnf*, s tím rozdílem, že v tomto případě jsou hodnoty a_{ii}, b_{ii}, c_{ii}, d_{ii}, a_{ss}, b_{ss}, c_{ss}, d_{ss} používané při výpočtu proměnné *vnf* extrahovány podle vzorce 5.6 (podkapitola 5.1.4), zatímco standardní klasifikace extrahuje tyto hodnoty dle vzorce 5.3 (podkapitola 5.1.3). Po výpočtu proměnné *vnf* optimalizovaná klasifikace pokračuje postupným optimalizovaným vyhodnocováním jednotlivých stupňů klasifikátoru. Pokud během tohoto postupného optimalizovaného vyhodnocování dojde k negativní detekci jakýmkoliv stupněm klasifikátoru, pak optimalizovaná klasifikace aktuální oblasti zájmu končí s výstupní hodnotou *negativní detekce*. Pokud během tohoto postupného optimalizovaného vyhodnocování dojde k pozitivní detekci všemi stupni klasifikátoru, pak optimalizovaná klasifikace aktuální oblasti zájmu končí s výstupní hodnotou *pozitivní detekce*.

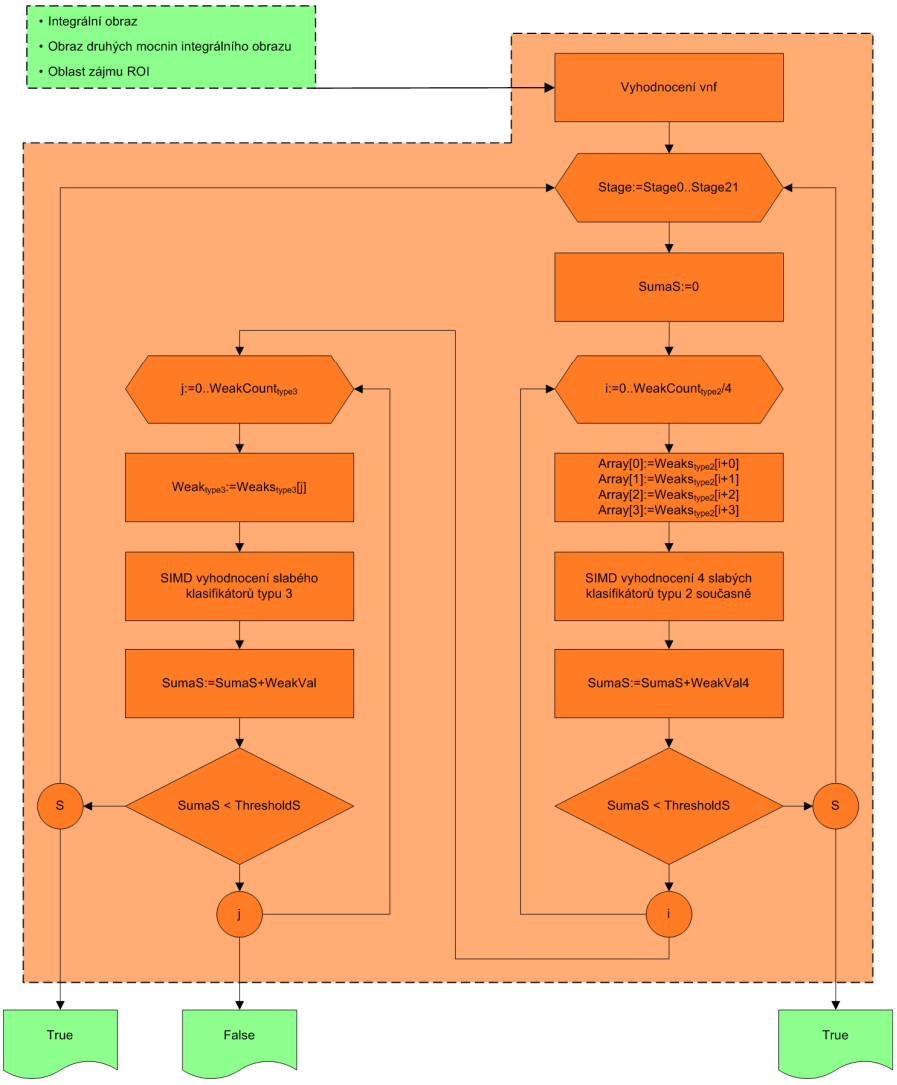
Optimalizované vyhodnocení jednoho stupně klasifikátoru lze rozdělit do 5 fází. První fáze začíná načtením proměnné *ThresholdS* dle souboru *haarcascade_frontalface_alt*, dále pokračuje vynulováním proměnné *SumaS* a poté dochází k naplněním fronty slabých klasifikátorů typu 2 a fronty slabých klasifikátorů typu 3 patřících k danému stupni klasifikátoru. Význam proměnných *ThresholdS* a *SumaS* byl definován v kapitole 5.1.3.

Na začátku druhé fáze je proveden test na počet položek ve frontě slabých klasifikátorů typu 2. Pokud je testovaný počet položek 0, pak optimalizované vyhodnocení jednoho stupně klasifikátoru pokračuje na začátek čtvrté fáze. Pokud je testovaný počet položek větší než 3, pak optimalizované vyhodnocení jednoho stupně klasifikátoru pokračuje načtením prvních čtyř položek z testované fronty do aktuální čtveřice slabých klasifikátorů typu 2 a dále pak zmíněné vyhodnocení pokračuje třetí fází. Pokud je testovaný počet položek v rozmezí od 1 do 3, tak dochází k načtení všech zbylých položek testované fronty do aktuální čtveřice a volná místa v této čtveřici jsou vyplněna samými 0 a poté optimalizované vyhodnocení jednoho stupně klasifikátoru pokračuje třetí fází.

Ve třetí fázi dochází k současnému vyhodnocení čtyř aktuálně načtených slabých klasifikátorů typu 2 prostřednictvím SIMD optimalizace na úrovni stupně klasifikátoru, viz kapitola 5.1.5.3. Výstupem tohoto souběžného vyhodnocení je hodnota *WeakVal4*. Proměnná *WeakVal4* je po jejím vrácení dále přičtena k proměnné *SumaS*. Poté dochází k porovnání proměnné *SumaS* s proměnnou *ThresholdS*. Pokud je *SumaS* větší rovna proměnné *ThresholdS*, tak optimalizované vyhodnocení daného stupně klasifikátoru končí *pozitivní detekcí*. Pokud je hodnota proměnné *SumaS* menší než hodnota proměnné *ThresholdS*, tak optimalizované vyhodnocení daného stupně klasifikátoru pokračuje na začátek druhé fáze.

Na začátku čtvrté fáze je proveden test na prázdnost fronty slabých klasifikátoru typu 3. Pokud testovaná fronta prázdná není, tak ukázková metoda pokračuje na začátek páté fáze., v opačném případě ukázková metoda optimalizovaného klasifikování končí *negativní detekcí*.

V páté fázi je provedeno optimalizované vyhodnocování aktuálně načteného slabého klasifikátoru typu 3, konkrétně optimalizované pomocí SIMD na úrovni slabého klasifikátoru, viz kapitola 5.1.5.2. Výstupem optimalizovaného vyhodnocení daného slabého klasifikátoru je hodnota *WeakVal*. Hodnota *WeakVal* je dále přičtena k proměnné *SumaS*. Hodnota proměnné *SumaS* je následně porovnána s hodnotou proměnné *ThresholdS*, pokud je *SumaS* větší nebo rovna hodnotě *ThresholdS*, pak optimalizované vyhodnocení jednoho stupně klasifikátoru končí *pozitivní detekci*, v opačném případě pokračuje na začátek čtvrté fáze.



Obrázek 5.11: Blokové schéma optimalizované klasifikace jednoho podokna.

Závěrem této kapitoly je vhodné poznamenat, že optimalizované klasifikování přináší průměrně cca 50% urychlení oproti standardnímu klasifikování při zachování stejné míry úspěšnosti klasifikování pro obrazy v rozlišení 800x600 s průměrným počtem tváří. Podrobné výsledky porovnání těchto dvou klasifikátorů je uveden v tabulce 7.2 v kapitole 7.

5.2 Návrh systému sledování tváří

Tato kapitola představuje návrh řešení sledování tváří ve videu. Uvedený návrh řešení je založen na používání Kalmanovi predikce (kapitola 4.10), dále na použití optimalizovaného detektoru tváří v obraze (kapitola 5.1.6) a na používání funkce *cvHaarDetectObject* z knihovny OpenCV.

Obecnou funkci sledování tváří ve videu lze rozdělit do 6 fází. Pro lepší pochopení obecné funkce je na obrázku 5.12 uvedeno její blokové schéma. První fáze je identická s první fází standardní obecné funkce pro detekování tváří ve videu, viz kapitola 5.1.

Druhá fáze začíná načtením prvního, inicializačního, vstupního snímku ze vstupní videosekvence. Dále je v inicializačním snímku provedena optimalizovaná detekce tváří v obraze, popsaná kapitolou 5.1.6. Výstupem této optimalizované detekce je pole tváří, podle něhož je naplněno pole detekovaných tváří a také pole sledovaných tváří. V obrázku 5.12 je pole detekovaných tváří označeno jako *poleDT* a pole sledovaných tváří je označeno jako *poleST*. Pole detekovaných tváří a pole sledovaných tváří pak mají své jednotlivé tváře uloženy na pozicích od 0 do *n-1*, kde *n* odpovídá velikosti pole vráceného optimalizovanou detekcí.

Třetí fáze začíná načtením vstupního snímku ze vstupní videosekvence a dále pokračuje vynulováním proměnné *index*. Poté popisovaná obecná funkce pokračuje na začátek 4 fáze.

Na začátku čtvrté fáze je proveden test na prázdnost pole sledovaných tváří, pokud je toto pole prázdné, tak obecná funkce sledování tváří přejde na začátek páté fáze. Pokud pole prázdné není, tak je jako aktuální tvář označena tvář z pole sledovaných tváří, jenž je v tomto poli uložena na pozici dané proměnnou *index*. Dále ve čtvrté fázi je provedena Kalmanova predikce pro aktuální tvář, což znamená, že z pozice levého horního rohu aktuální tváře je předpověděna nová pozice levého horního rohu aktuální tváře. Při ideální Kalmanově predikci by předpověděná pozice měla přesně odpovídat pozici levého horního rohu aktuální tváře v načteném vstupním snímku. Čtvrtá fáze dále pokračuje vymezením obdélníkové oblasti pro následnou cílenou detekci. Vymezená oblast je určena podle následující vzorce:

$$\begin{aligned} w_o &= 1.65 \cdot w_t \\ h_o &= 1.45 \cdot h_t \\ x_o &= \min\left(0, x_p - \frac{w_o - w_t}{2}\right), \\ y_o &= \min\left(0, y_p - \frac{h_o - h_t}{2}\right) \end{aligned} \qquad (5.8)$$

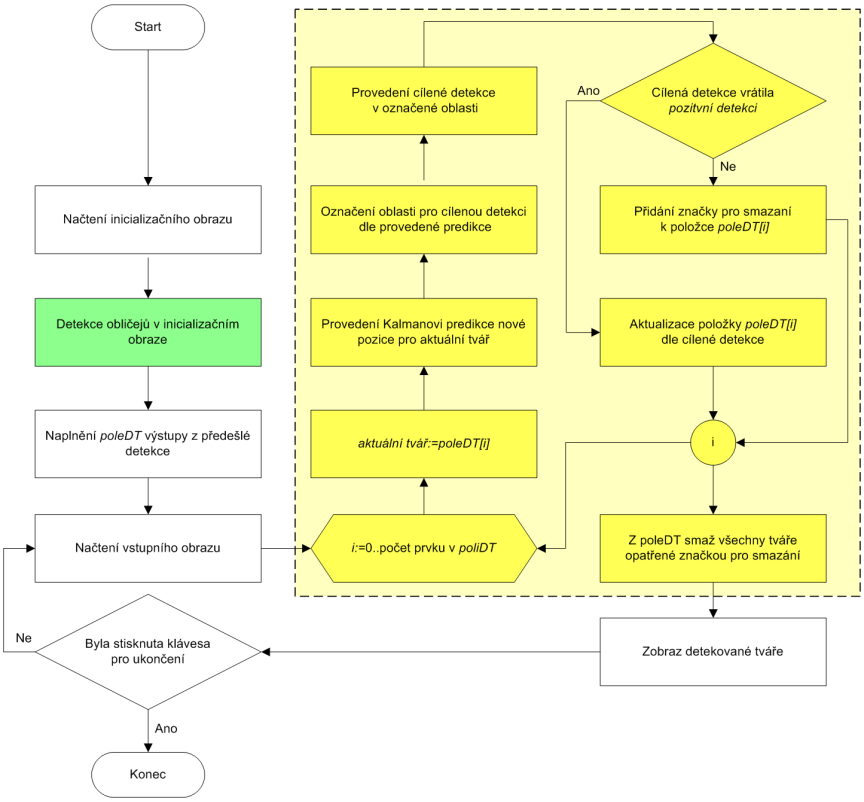
kde w_o, resp. h_o odpovídá šířce, resp. výšce vymezené oblasti, x_o, resp. y_o odpovídá *x-ové*, resp. *y-ové* souřadnici levého horního rohu vymezené oblasti, x_p, resp. y_p odpovídá *x-ové*, resp. *y-ové* souřadnici predikované pozice levého horního rohu aktuální tváře a w_t, resp. h_t odpovídá šířce, resp. výšce aktuální tváře. Je vhodné poznamenat, že hodnoty 1.65 a 1.45 z předešlého vzorce byly nalezeny experimentálně a jsou považovány jako optimální pro účely této diplomové práce. Tyto hodnoty bylo nutné zvolit tak, aby vymezená oblast byla dostatečně rozsáhlá na to, aby byla eliminována nepřesnost predikce a současně, aby vymezená oblast nebyla zbytečně moc rozsáhlá a nesnižovala tak efektivitu následné cílové detekce, jelikož čím větší rozsah vymezené oblasti, tím více *negativních detekcí* musí následná cílená detekce provést.

Po dokončení výpočtu vymezené oblasti následuje již několikrát zmiňovaná cílená detekce prováděná nad touto oblastí. Cílená detekce je realizovaná funkcí *cvHaarDetectObject* z knihovny OpenCV. Vstupem cílené detekce je vymezená oblast a výstupem je jedna tvář v případě úspěšného nalezení sledované tváře, nebo hodnota *negativní detekce* v opačném případě. Pro maximální

efektivitu je předešlá funkce volána s nastavenou volbou *CV_HAAR_FIND_BIGGEST_OBJECT*, což způsobí, že funkce *cvHaarDetectObject* hledá pouze jednu, největší, tvář ve vymezené oblasti, což je mnohem rychlejší než standardní volání této funkce. Z předešlé věty vyplývá další skutečnost, že pokud by cílená detekce byla prováděna nad celým snímkem, tak by pro pět tváří byla nalezena pětkrát ta samá, největší, tvář. Správnou volbou vymezené oblasti podle výše uvedeného vzorce 5.8 však k tomuto jevu nedochází. Pokud funkce *cvHaarDetectObject* ve vymezené oblasti detekuje tvář, je tato tvář vrácena jako výsledek cílené detekce. Vrácenou tváří je poté nahrazena tvář v poli sledovaných tváří na pozici dané proměnnou *index*. Pokud funkce *cvHaarDetectObject* ve vymezené oblasti žádnou tvář nenalezne, je výstupní hodnotou cílové detekce hodnota *negativní detekce*. Při vrácení negativní detekce následuje přidání značky pro smazání, jak k tváři v poli sledovaných tváří na pozici dané proměnnou *index*, tak k tváři v poli detekovaných tváří rovněž na pozici dané proměnnou *index*. Pokud je hodnota proměnné *index* menší než velikost pole, tak je tato proměnná dále inkrementována a poté následuje skok na začátek čtvrté fáze, v opačném případě obecná funkce sledování tváří ve videu pokračuje na začátek páté fáze.

V páté fázi dochází k postupnému smazání všech tváří z pole detekovaných tváří a z pole sledovaných tváří, jež jsou opatřeny značkou pro smazání. Po smazání příslušných položek z obou polí pokračuje obecná funkce sledování tváří na začátek fáze šest.

V šesté fázi dochází k zobrazení všech tváří z pole sledovaných tváří. Poté je vyhodnoceno, zda byla stisknuta klávesa pro ukončení programu, pokud ano, obecná funkce sledování tváří ve videu končí, v opačném případě tato funkce pokračuje na začátek třetí fáze.



Obrázek 5.12: Blokové schéma sledování tváří ve videu.

Závěrem této podkapitoly je vhodné poznamenat, že uvedená obecná funkce sledování tváří ve videu funguje správně, pouze pokud je počet tváří ve scéně konstantní a jejich pohyb přirozený, resp. dobře předvídatelný po celou dobu videosekvence. O tom jak se vypořádat s videosekvencemi, v nichž se počet tváří mění, nebo v nichž je pohyb tváří hůře předvídatelný, je psáno v následující kapitole.

5.3 Návrh systému souběžného sledování a detekce tváří

Tato kapitola představuje návrh řešení, jež je založený na běhu ve více vláknech, kdy v jednom vlákně probíhá sledování tváří ve videu, dle obecné funkce představené v kapitole 5.2 a ve druhém vlákně současně probíhá optimalizovaná detekce tváří ve videu, dle obecné funkce, jejíž princip byl popsán v rámci kapitoly 5.1.6.

Ještě před popisem obecné funkce realizující souběžné sledování a detekování tváří je vhodné zmínit, že samotným sledováním tváří je sice možné detekovat zmizení jedné ze sledovaných tváří ze scény, avšak touto metodikou není možné detekovat výskyt nové tváře ve scéně. Detekci výskytu nové tváře ve scéně lze provést optimalizovanou detekcí tváří nad celým obrazem. Zmíněná optimalizovaná detekce nad celým vstupním snímkem po zbytek této práce ponese označení *kontrolní detekce*, jelikož svou činností v podstatě provádí kontrolu stavu ve scéně. Zmíněný celý vstupní snímek, nad nímž je *kontrolní detekce* prováděna, pak bude po zbytek této práce analogicky označován jako *kontrolní snímek*. Na základě výstupu z *kontrolní detekce* je vždy aktualizováno pole sledovaných tváří, a tím je odstraněna nevýhoda předešlého řešení prezentovaného v kapitole 5.2.

Obecnou funkci realizující prezentovaný návrh lze rozdělit do 6 fází. Blokové schéma zmíněné obecné funkce je uvedeno na obrázku 5.13. První dvě fáze jsou naprosto identické s prvními dvěma fázemi obecné funkce sledování tváří, jež byly popsány v rámci kapitoly 5.2. Je vhodné poznamenat, že první dvě fáze probíhají v jednom, hlavním, vlákně popisované obecné funkce. Po dokončení prvních dvou fází dochází k rozběhnutí nového, vedlejšího, vlákna. Pro účely této kapitoly je hlavní vlákno nazváno jako sledovací vlákno a vedlejší vlákno jako detekční vlákno.

Po rozběhnutí vedlejšího vlákna dále popisovaná obecná funkce pokračuje v hlavním vlákně třetí, čtvrtou a pátou fází, jež jsou naprosto shodné s třetí, čtvrtou a pátou fází obecné funkce sledování tváří ve videu. Je vhodné poznamenat, že tyto tři fáze v obecné funkci sledování tváří realizují sledování tváří na základě pole sledovaných tváří. Stěžejní rozdíl mezi popisovanou obecnou funkcí a obecnou funkcí sledování tváří ve videu je v plnění pole sledovaných tváří, kdy v obecné funkci sledování tváří je pole sledovaných tváří plněno pouze v jednom místě a to po detekci ve druhé fázi obecné funkce sledování tváří. V popisované obecné funkci však dochází k plnění pole sledovaných tváří na dvou místech a to po detekci ve druhé fázi popisované obecné funkce a navíc

také po *kontrolní detekci* prováděné ve vedlejším vlákně. Průběh popisované obecné funkce ve vedlejším vlákně bude popsán v následujícím odstavci. Popisovaná funkce v hlavním vlákně dále pokračuje šestou fází, jež se od šesté fáze obecné funkce sledování tváří ve videu liší pouze v chování po pozitivním vyhodnocení stisku ukončovací klávesy. V obecné funkci sledování tváří je po pozitivním vyhodnocení stisku ukončovací klávesy obecná funkce sledování tváří ihned ukončena, zatímco v popisované funkci je po stejné události obecná funkce ukončena s prodlevou. Zmíněná prodleva je způsobena čekáním na ukončení vedlejšího vlákna.

Vedlejší vlákno popisované obecné funkce cyklicky provádí *kontrolní detekci* tváří v načteném *kontrolním snímku*. Výsledek zmíněné *kontrolní detekce* je pak dále předán metodě provádějící aktualizaci pole sledovaných tváří. Je vhodné zdůraznit, že v rámci této aktualizace mimo jiné probíhá i načtení nového *kontrolního snímku*. Nakonec dochází k vyhodnocení stisku ukončovací klávesy. Tyto tři akce, *kontrolní detekce*, aktualizace a vyhodnocení, jsou cyklicky prováděny, dokud vyhodnocení stisku ukončovací klávesy končí negativně. Po pozitivním vyhodnocení stisku ukončující klávesy vedlejší vláno končí. V dalším odstavci bude podrobněji popsán průběh aktualizace pole sledovaných tváři.

Aktualizace pole sledovaných tváří probíhá v 6 fázích. Je vhodné zmínit, že bloky spojené s aktualizací sledovaných tváří jsou v obrázku 5.13 umístěny v modré oblasti. V první fázi aktualizace sledovaných tváří dochází k naplnění dočasného pole nově detekovaných tváří dle výsledků předcházející *kontrolní detekce*. Je vhodné poznamenat, že pole nově detekovaných tváří je v obrázku 5.13 označováno jako *poleNDT*.

Ve druhé fázi dochází k postupnému průchodu polem detekovaných tváři, kdy je pro každou položku *p* tohoto pole, proveden pokus o vyhledání její kopie v dočasném poli nově detekovaných tváří. Pokud je pokus úspěšný, je vyhledaná kopie odstraněna z dočasného pole nově detekovaných tváři. Pokud pokus úspěšný není, tak je extrahována pozice *i* položky *p* v poli detekovaných tváři, poté dochází k odstranění tváře z pole sledovaných tváří, jež se v tomto poli vyskytuje na pozici *i*.

Ve třetí fázi dochází k přidání všech položek z dočasného pole detekovaných tváří do pole sledovaných tváři. Dále v této fázi následuje smazání celého pole detekovaných tváří. Třetí fáze končí načtením nového *kontrolního snímku* ze vstupní videosekvence.

Čtvrtá a pátá fáze aktualizace probíhá takřka stejně jako čtvrtá a pátá fáze obecné funkce sledování tváří. Jediným rozdílem v provádění těchto fází je, že čtvrtá a pátá fáze popisované aktualizace pracuje s *kontrolním snímkem*, zatímco čtvrtá a pátá fáze obecné funkce sledování tváří pracuje se standardním vstupním snímkem. Čtvrtá a pátá fáze popisované aktualizace pak mění obsah pole sledovaných tváří podle aktuálního *kontrolního snímku* způsobem popsaným v kapitole 5.2.

V šesté závěrečné fázi pak už jen dochází k přidání všech položek z pole sledovaných tváři do pole detekovaných tváři.

Obrázek 5.13: Blokové schéma souběžné detekce a sledování tváří ve videu, modrá oblast ohraničuje bloky vyhodnocované v rámci aktualizace pole sledovaných tváří.

5.4 Návrh systému používajícího buffer

Tato kapitola představuje návrh řešení, jež je stejně jako předchozí navržené řešení, založené na běhu ve více vláknech, kdy v jednom vlákně probíhá sledování tváří ve videu, dle obecné funkce představené v kapitole 5.2 a ve druhém vlákně probíhá současně optimalizovaná detekce tváří ve videu, dle obecné funkce, popsané v kapitole 5.1.6.

Popisovaný návrh řešení je pro lepší pochopení také prezentován formou zjednodušeného blokového schématu, viz obrázek 5.14 uvedený na konci této podkapitoly. Zmíněný návrh přináší především buffrovanou vstupní video sekvenci. Konkrétně buffruje *n* sekund vstupní videosekvence, což způsobí *n* sekundovou prodlevu oproti vysílané vstupní video sekvenci. Zmíněná prodleva je určena dobou, za kterou je provedena optimalizovaná detekce obličejů v jednom vstupním obraze.

Zmíněná prodleva je závislá na použitém procesoru, proto je vždy před startem hlavní smyčky explicitně určena. Zmíněná prodleva snižuje přirozený vjem uživatele, za předpokladu, že je navržené řešení aplikováno na detekci obličejů ve scéně, v níž se uživatel vyskytuje. Protože pokud aplikace detekuje obličeje ve scéně, jejíž součástí je i sám uživatel, pak uživatel vidí na monitoru dění, jež je o *n* zpožděné oproti dění, jež vnímá ve skutečnosti.

Naproti tomu, pokud je aplikace nasazena mimo scénu, v níž se vyskytuje uživatel, např. dopravní kamery, pak zmíněný problém nenastává. Další vhodný prostor pro uplatnění tohoto návrhu je detekce obličejů v dříve vytvořeném video souboru.

Buffrovaná vstupní videosekvence v popisovaném navrženém řešení způsobí, že každý snímek získaným ze vstupní videosekvence v čase *t*, bude zobrazován až v čase $t+n$. Z předchozí věty plyne, že pokud je odstartována *kontrolní detekce* hned v čase získání *kontrolního snímku*, tj. v čase t_{ks}, tak v čase zobrazení *kontrolního snímku*, tj. v čase $t_{ks}+n$, bude *kontrolní detekce* již dokončena. Předchozí věta jinými slovy znamená, že při používání bufferu je možné v čase zobrazení *kontrolního snímku* plynule zobrazit tento snímek i s výsledky *kontrolní detekce*, bez čekání na dokončení *kontrolní detekce*. V čase t_{ks} je zobrazen minulý *kontrolní snímek* pořízený v čase $t_{ks}-n$. V časovém intervalu od t_{ks} do $t_{ks}+n$, resp. během provádění nové *kontrolní detekce* dochází ke dvěma činnostem. První činností je zobrazení snímků pořízených v intervalu od $t_{ks}-n$ do t_{ks} a následné označení tváří na základě sledování tváří detekovaných v *kontrolním snímku* pořízeném v čase $t_{ks}-n$. Ve druhé je ukládání snímků do bufferu získaných ze vstupní videosekvence v intervalu od t_{ks} do $t_{ks}+n$.

Je vhodné poznamenat, že pokud bychom nepoužili buffer, pak by platilo, že všechny snímky by byly pořizovány i zobrazovány ve stejném čase. *Kontrolní snímek* by tak byl pořízený v čase t_{ks} a by byl zobrazen v témže čase t_{ks}. Výpočet jedné *kontrolní detekce* však trvá stále *n* sekund. Z předchozí věty plyne, že při zobrazení *kontrolního snímku* by *kontrolní detekce* nad tímto snímkem ještě nebyla dokončena. Jedinou možností by tak bylo zobrazit *kontrolní snímek* se zvýrazněnými výsledky dle předchozí *kontrolní detekce*, tedy dle *kontrolní detekce* provedené nad *kontrolním snímkem* pořízeným v čase $t_{ks}-n$. Což by vedlo k zobrazování *n* sekund starých výsledků.

Návrh 5.3 sice prezentuje možnost současného použití detekování i sledování tváří bez bufferu, ale nepoužití bufferu je vykoupeno neustálým porovnáváním výsledků *kontrolní detekce* s výsledky ze souběžného sledování a na základě tohoto porovnání je dále neustále prováděna aktualizace obou polí. Návrh využívající buffer se ale těmto porovnáním a aktualizacím za cenu *n* sekundového zpoždění oproti vstupní video sekvenci vyhne, což je hlavní důvod pro vytvoření tohoto návrhu.

Návrh s bufferem pak značně vyplývá z předchozích odstavců, a lze ho popsat v šesti fázích. V první fázi dochází k načtení vstupního klasifikátoru pro cílenou detekci a pro *kontrolní detekci*. Poté dochází k naplnění vstupního bufferu ze vstupní videosekvence. Dále pak dochází ke *kontrolní detekci* nad prvním snímkem z bufferu. Poté následuje explicitní určení *n* sekundové časové prodlevy, podle doby trvání předchozí *kontrolní detekce*, dle následujícího vzorce

$$n = \Delta T + tolerance, \tag{5.9}$$

kde symbol ΔT značí trvání předchozí *kontrolní detekce* a symbol tolerance značí přidanou toleranci vůči změnám ΔT. Je vhodné poznamenat, že změny ΔT pramení ze skutečnosti, že *kontrolní detekce* jsou prováděny nad různými *kontrolními snímky*, přičemž tyto změny bývají velice malé (řádově desetiny procent). Na konci první fáze dochází k naplnění pole detekovaných a sledovaných tváří podle předešlé *kontrolní detekce*.

Ve druhé fázi dochází k rozběhnutí vedlejšího vlákna, ve kterém je prováděna pouze operace *kontrolní detekce* nad posledním snímkem v bufferu. Zmíněná *kontrolní detekce* se ve vedlejším vlákně neustále opakuje až do té doby, dokud není detekován stisk klávesy pro ukončení programu. Po detekci stisknutí ukončovací klávesy je vedlejší vlákno ukončeno. Dále program pokračuje v hlavním vlákně přechodem do třetí fáze.

Ve třetí fázi program postupně pro každý snímek *p* z bufferu, s výjimkou posledního snímku provede proces složený ze třech kroků. Prvním krokem tohoto procesu je predikce pozic všech sledovaných tváří ve snímku *p*. Ve druhém kroku následuje zakreslení všech predikovaných pozic do snímku *p*. Ve třetím kroku je dále upravený snímek *p* zobrazen. Ve čtvrtém kroku je do snímku *p* uložen aktuální snímek ze vstupní videosekvence. Zmíněný 4 krokový proces pro každý snímek *p* trvá *m* sekund, kdy *m* je dáno následujícím vzorcem:

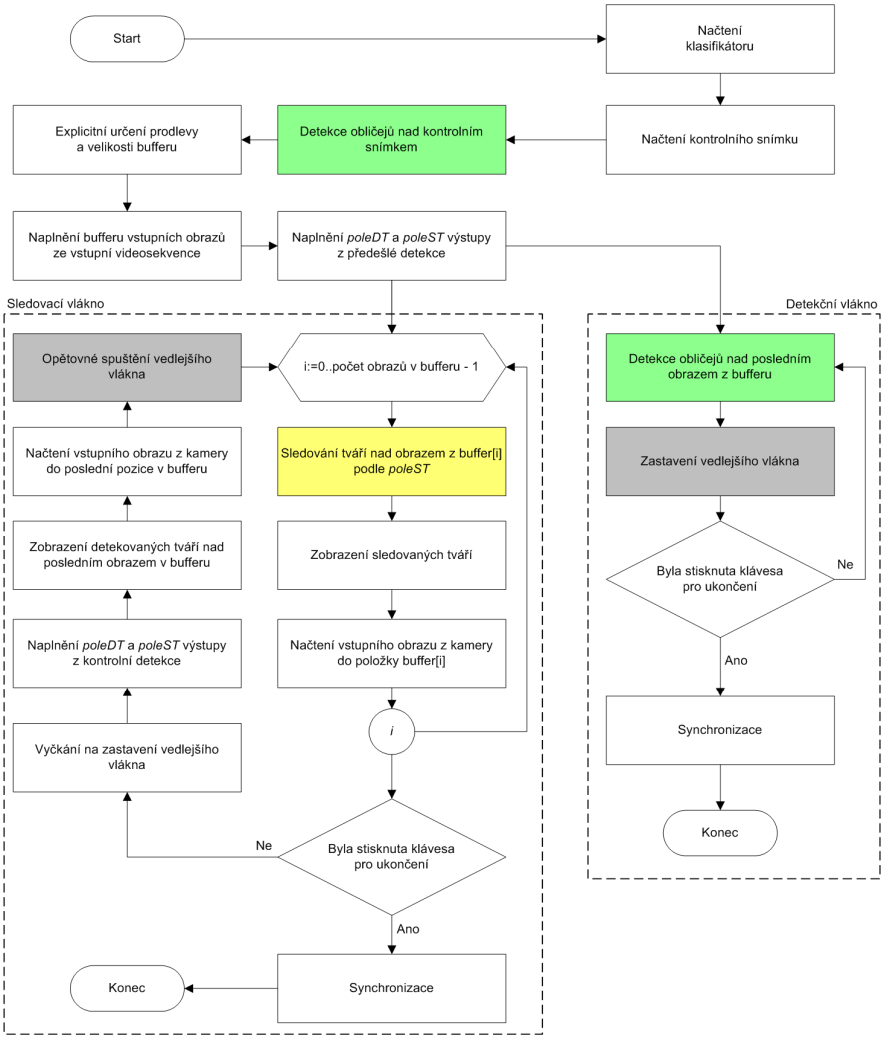
$$m = \frac{n}{size}, \tag{5.10}$$

kde *size* značí počet položek, resp. snímků v bufferu, defaultně je tento počet roven 20, značka *n* je dána vzorcem 5.9.

Na začátku čtvrté fázi je vyhodnoceno, zda již byla během programu stisknuta klávesa pro ukončení. Pokud klávesa stisknuta byla, program pokračuje na začátek šesté fáze. V opačném případě pokračuje vyčkáním na dokončení *kontrolní detekce*. Pokud se program nikde nezasekl, tak toto čekání je nulové. Následuje naplnění pole sledovaných tváří a pole detekovaných tváří podle výsledků z *kontrolní detekce*. Dále program pokračuje zakreslením detekovaných tváří do posledního snímku z bufferu, a poté je takto upravený snímek zobrazen. Po zobrazení následuje načtení aktuálního snímku ze vstupní videosekvence do posledního snímku v bufferu.

V páté fázi je opět spuštěna *kontrolní detekce* nad posledním snímkem v bufferu, a dále pak program v hlavním vlákně pokračuje na začátek třetí fáze.

V šesté fázi je vyčkáno na ukončení *kontrolní detekce*, resp. na ukončení celého vedlejšího vlákna, a poté je ukončen i celý program.



Obrázek 5.14 Blokové schéma souběžné detekce a sledování tváří ve videu s použitím buffrované vstupní videosekvence.

6 Praktická implementace

V úvodu této kapitoly je vhodné zmínit, že realizovaná knihovna je napsána v jazyce C++. Realizovaná knihovna ve své implementaci využívá knihovnu tixml pro parsování vstupního xml klasifikátoru, jenž je převzat z knihovny OpenCV. Zmíněná knihovna dále využívá i samotnou knihovnu OpenCV, z níž čerpá především funkce pro načítání vstupní videosekvence, funkce pro zobrazování výsledků a funkce použité během cílené detekce.

V dalším průběhu této kapitoly bude popsáno programové rozhraní této knihovny, a také budou popsány dva z možných způsobů, jak tuto knihovnu implementovat do vlastní aplikace, konkrétně aplikace pro real-time detekci/sledování obličejů ve videu.

Na závěr této kapitoly je pak uvedeno zhodnocení implementovaného řešení, konkrétně potom zhodnocení dosažené úspěšnosti a rychlosti při detekování a sledování tváři ve videu.

6.1 Programové rozhraní realizované knihovny

Tato kapitola popisuje programové rozhraní implementované knihovny, kdy jsou uvedeny pouze obecné principy funkcí, jež tvoří dané programové rozhraní. Tento způsob byl zvolen s ohledem na skutečnost, že podrobný popis všech funkcí by byl přehnaně zdlouhavý. A dále s ohledem na skutečnost, že při psaní zdrojových kódů byla snaha psát přehledný a dostatečně komentovaný kód, proto by neměl být problém v případě potřeby obrátit se právě na zdrojové kódy realizovaného řešení.

Tato kapitola je dále rozdělena do tří podkapitol, kdy první popisuje programové rozhraní pro detekci lidské kůže ve videu, druhá popisuje programové rozhraní pro detekci obličejů ve videu a konečně třetí popisuje programové rozhraní pro sledování obličejů ve videu.

Ještě před samotným popisem jednotlivých rozhraní je vhodné ujasnit si rozdíl mezi pojmem metoda a pojmem funkce. Tyto dva pojmy se liší ve výstupní hodnotě, kdy metoda výstupní hodnotu nemá, zatímco funkce výstupní hodnotu má.

6.1.1 Programové rozhraní pro detekci kůže

Toto rozhraní je tvořeno metodami `gm_prepareGauss` a `gm_skinDetection`.

Metoda `gm_prepareGauss` je používána k inicializaci pravděpodobnostní matice z trénovacího obrázku, viz kapitola 3.4, přičemž pro zmíněnou pravděpodobnostní matici je vytvořena vlastní struktura `TGaussStruct`.

Metoda `gm_skinDetection` slouží k detekci lidské kůže v obraze. Tato metoda umožňuje 3 způsoby detekce lidské kůže a 4 režimy zvýraznění detekovaných oblastí vstupního obrazu. Volba způsobu detekce je závislá na vstupním parametru `typeOfSkinDetection` a volba zvýraznění

detekovaných oblastí je řízena parametrem `typeOfSkinClasification`. Oba parametry lze libovolně kombinovat.

První parametr je typu `int` a nabývá hodnot 0, 1, 2. Hodnota 0 odpovídá detekci popsané v kapitole 3.4, z čehož plyne, že tento způsob detekce vyžaduje korektně inicializovanou pravděpodobnostní matici, z čehož dále plyne, že v tomto případě musí detekční metodě nutně předcházet metoda `gm_prepareGauss`, jež pravděpodobnostní matici korektně inicializuje. Další volbou parametru `typeOfSkinDetection` je hodnota 1, resp. 2, jež signalizuje detekci na základě *RGB*, resp. YC_RC_B popsanou v kapitole 3.1.

Druhý parametr je rovněž typu `int` a nabývá rovněž hodnot 0, 1, 2. Hodnotou 0, resp. 1, resp. 2 je dosaženo způsobu zvýrazňování kůže, jež je znázorněn na obrázku 3.4-b, resp. 3.4-c, resp. 3.5-c.

6.1.2 Programové rozhraní pro detekci obličejů

Toto rozhraní tvoří metody `gm_prepare_CV_FaceDetector` a `gm_prepare_MY_FaceDetector`, funkce `gm_CV_FaceDetector`, `gm_CV_FaceReFind`, `gm_MY_FaceDetector`, `gm_MY_FaceReFind`, `SSEAccelerationSupported` a třídy `gm_Classifier`, `gm_Stage`, `gm_Weak`, `gm_Rect`.

Metoda `gm_prepare_CV_FaceDetector` je používána k inicializaci klasifikátoru pro účely OpenCV detekční funkce `cvHaarDetectObjects`. Tento klasifikátor je inicializován podle vstupního souboru `haarcascade_frontalface_alt.xml`

Metoda `gm_prepare_MY_FaceDetector` je používána k inicializaci klasifikátoru `gm_Classifier`. Tento klasifikátor je využíván funkcemi začínajícím předponou `gm_MY`. Inicializace tohoto klasifikátoru je pak stejně jako v předchozím případě prováděna na základě vstupního souboru `haarcascade_frontalface_alt.xml`.

Funkce `gm_CV_FaceDetector` provádí detekci obličejů v obraze s využitím OpenCV funkce `cvHaarDetectObjects`. Z předchozí věty plyne, že před první detekcí provedenou funkcí `gm_CV_FaceDetector` je nutné korektně inicializovat vstupní klasifikátor pomoci funkce `gm_prepare_CV_FaceDetector`. Výstupní hodnotou funkce `gm_CV_FaceDetector` je počet detekovaných tváří v obraze. Nalezené tváře tato funkce vrací pomocí kontejneru `detected_faces`, jenž je do funkce předáván jako vstupní parametr s referencí. Zmíněný kontejner je naplněn obdélníky, kdy každý jeden obdélník ohraničuje jednu nalezenou tvář v obraze. Reálná implementace zmíněného kontejneru je pak konkrétně `std::vector<CvRect>`.

Funkce `gm_CV_FaceReFind` provádí cílenou detekci obličejů v obraze, resp. detekci největšího obličeje ve vymezené oblasti vstupního obrazu, kdy princip a účel cílené detekce jsou blíže popsány v kapitole 5.2. Tato funkce funguje na stejném základě jako předchozí funkce, z čehož plyne, že před prvním zavoláním této funkce je rovněž nutné korektně inicializovat vstupní klasifikátor pomocí funkce `gm_prepare_CV_FaceDetector`. Výstupní hodnotou této funkce je `TRUE` v případě,

že cílená detekce ve vymezené oblasti našla tvář, v opačném případě je výstupní hodnotou hodnota FALSE. Vymezená oblast cílené detekce je této funkci předána jako obdélník typu CvRect.

Funkce gm_MY_FaceDetector a gm_MY_FaceReFind plní stejný účel jako názvem odpovídající funkce s předponou gm_CV a můžeme tak od nich očekávat identické výstupy jako od odpovídajících funkcí s předponou gm_CV. Detekčních funkce s předponou gm_MY však detekují na základě klasifikátoru typu gm_Classifier v porovnání s detekčními funkcemi s předponou gm_CV, jež provádějí detekce na základě funkce cvHaarDetectObjects. Detekční funkce gm_MY umožňují 4 režimy detekce, kdy volba režimu probíhá na základě hodnoty vstupního parametru sse_support. Zmíněný parametr je typu int a nabývá hodnot od 0 do 3. Hodnota 0 signalizuje režim standardní detekce popsané v kapitole 5.1.3 s optimalizovaným procházením podoken, jež bylo popsáno v kapitole 5.1.4. Hodnota 1 signalizuje režim standardní detekce s využitím optimalizace pomocí SIMD na úrovni obdélníka. Hodnota 3 signalizuje režim optimalizované detekce popsané v kapitole 5.1.6. Hodnota 2 signalizuje režim optimalizované detekce, jenž se vyhýbá používání specifických globálních proměnných. Tyto globální proměnné způsobují rychlejší běh detektoru, naproti tomu je ale jejich použitím zabráněno souběžnému běhu více gm_MY funkcí ve více vláknech. Je vhodné poznamenat, že návrh prezentovaný kapitolou 5.3 lze provést i s parametrem sse_support nastaveným na hodnotu 3, jelikož v jednom vlákně běží gm_MY detektor zatímco ve druhém vlákně běží gm_CV detektor, což není situace, kdy běží dvě gm_MY funkce současně.

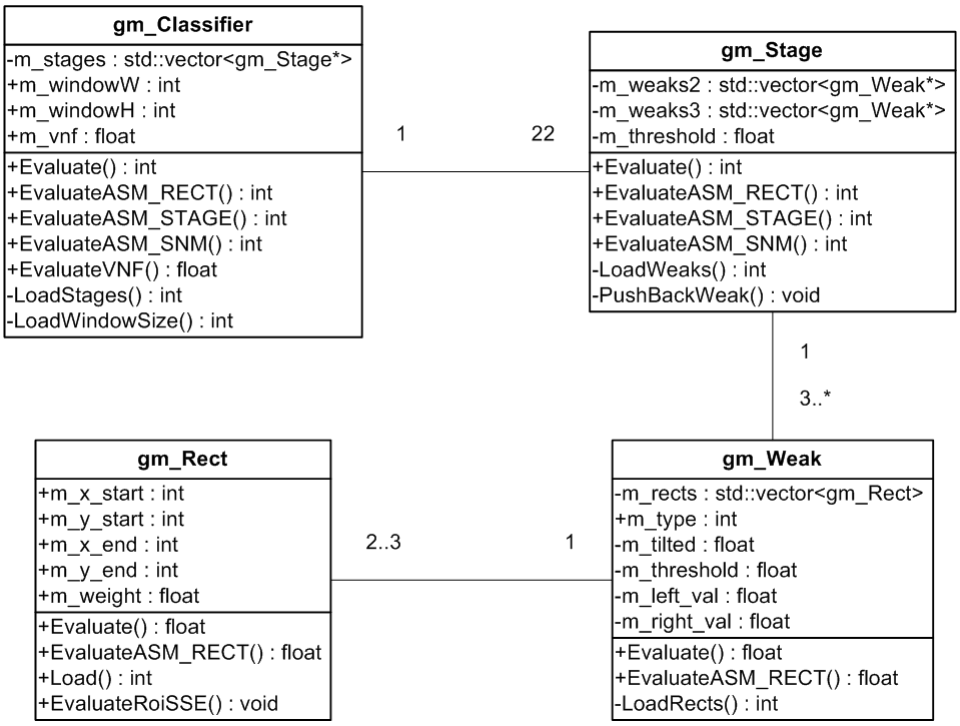
Funkce SSEAccelerationSupported testuje, zda je program spuštěn na PC vybaveným SIMD technologiemi nutnými pro optimalizovanou detekci. Tato funkce vrací 0, resp. 1 pokud PC SIMD nepodporuje, resp. podporuje.

V úvodu zmíněné třídy jsou uspořádány v hierarchii, kdy nejvýše postavenou třídou je gm_Classifier. Tato hierarchie kopíruje uspořádání použité ve vstupním souboru, podle něhož je třída gm_Classifier inicializována. Hierarchii a strukturu zmíněných tříd nejlépe popíše zjednodušený třídní diagram, jenž je uveden na obrázku 6.1. Ve zmíněném zjednodušeném třídním diagramu je vhodné blíže představit funkce s předponou Load a funkce s předponou Evaluate.

Funkce začínající předponou Load načítají příslušné třídní atributy extrahováním patřičných hodnot ze vstupního souboru haarcascade_frontalface_alt.xml. Principy zmíněné extrakce vyplývají ze struktury klasifikátoru zmíněné v kapitole 5.1.1. Pro extrahování je využito funkcí z knihovny tixml. Funkce LoadWindowSize konkrétně načte šířku, resp. výšku podokna klasifikátoru do proměnné m_windowW, resp. m_widnowH daného klasifikátoru. Funkce LoadStages poté načte všechny stupně klasifikátoru. V rámci načtení jednoho stupně klasifikátoru je načtena jedna hodnota m_stage_threshold a dále zavolána funkce LoadWeaks. Funkce LoadWeaks načte všechny slabé klasifikátory k danému jednomu stupni načítaného klasifikátoru. V rámci načítání jednoho slabého klasifikátoru je dále volána funkce LoadRects. Funkce LoadRects načte všechny obdélníky k danému jednomu slabému klasifikátoru. Počet načtených obdélníků je dále uložen jako

hodnota atributu m_type daného slabého klasifikátoru. Zbylé atributy plněného slabého klasifikátoru, viz obrázek 6.1, jsou načteny podle výše zmíněného vstupního souboru. V rámci načtení jednoho obdélníka dojde k naplnění všech jeho atributů, viz obrázek 6.1. Atribut `m_weight` je dále vydělen součinem 20·20, resp. součinem šířky a výšky podokna, viz vzorec 5.4 uvedený v kapitole 5.1.3.

Funkce začínající předponou `Evaluate` provádějí vyhodnocení na různých úrovních klasifikátoru. Tyto funkce se dělí do 4 skupin, kdy tyto skupiny reflektují volbu výše zmíněného parametru `sse_support`, jenž vstupuje do `gm_MY` detekčních funkcí. První, resp. druhá, resp. třetí, resp. čtvrtá skupina funkcí je volána v případě parametru `sse_support` nastaveného na 0, resp. 1, resp. 2, resp. 3. Do první skupiny pak konkrétně patří funkce tvořené pouze názvem `Evaluate` bez přípony. Do druhé skupiny patří funkce tvořené názvem `EvaluateASM_RECT`. Do třetí skupiny patří funkce s názvem `EvaluateASM_STAGE`. Funkce s předponou `Evaluate` ze třídy `gm_Classifier`, resp. `gm_Stage`, resp. `gm_Weak`, resp. `gm_Rect` provádí vyhodnocení celého klasifikátoru, resp. jednoho stupně klasifikátoru, resp. jednoho slabého klasifikátoru, resp. jednoho obdélníka. Princip zpracování výstupní hodnoty jednoho obdélníka, jednoho slabého klasifikátoru, jednoho stupně klasifikátoru za účelem vyhodnocení celého klasifikátoru pro všechny volby parametru `sse_support` byl popsán v 5. kapitole a proto tady nebude opakován.



Obrázek 6.1: Zjednodušený třídní diagram použitého klasifikátoru

6.1.3 Programové rozhraní pro sledování obličejů

Toto rozhraní tvoří funkce `gm_CompareFaces`, metoda `gm_AddTrackFaces` a třídy `gm_Face` a `gm_KalmanFilter`. Zmíněné třídy jsou také popsány formou zjednodušeného třídního diagramu na obrázku 6.2 uvedeného v závěru této podkapitoly.

Funkce `gm_CompareFaces` provádí porovnání pole sledovaných tváří s polem nově detekovaných tváří. Pole sledovaných tváří je implementováno jako `std::vector<gm_Faces*>`, pole nově detekovaných tváří je implementováno jako `std::vector<CvRect>`. Popisovaná funkce pro každou sledovanou tvář zavolá její třídní funkci `FoundInDetectedArray`, kde je jako vstupní parametr této funkce použito pole nově detekovaných tváří. Podle výstupní hodnoty zmíněné funkce dojde buď k odstranění sledované tváře z pole sledovaných tváří, nebo k odstranění detekované tváře z pole nově detekovaných tváří. Volba mazané položky vychází z principu popsaného v kapitole 5.3.

Metoda `gm_AddTrackFaces` vloží pro každou položku z pole nově detekovaných tváří odpovídající záznam do pole sledovaných tváří.

Třída `gm_KalmanFilter` je používána pro předvídání pozice sledovaného objektu ve snímku *n+1* ze snímku *n*. Zmíněná predikce je založena na Kalmanově filtru, jenž byl popsán v kapitole 4.10. Z kapitoly 4.10 plyne, že Kalmanův filtr je tvořen dvěma fázemi, předpovědí a korekcí. První, resp. druhá fáze je ve třídě `gm_KalmanFiltr` realizována funkci `predict`, resp. `correct`. Třída `gm_KlmanFiltr` je dále složena z řady atributů typu matice, kdy význam těchto atributů byl rovněž definován v kapitole 4.10. O korektní inicializaci těchto atributů se stará metoda `init`. Třída `gm_KalmanFiltr` je pak v realizovaném řešení používána zejména v kombinaci se třídou `gm_Face`, kdy každá jedna instance třídy `gm_Face` obsahuje referenci na jednu unikátní instanci třídy `gm_KalmanFilter`. Unikátní instance třídy `gm_KalmanFilter` je vytvořena v rámci tvorby instance třídy `gm_Face`. Instance třídy `gm_Face` pak využívá přidruženou instanci třídy `gm_KalmanFilter` pro předpověď své nové pozice ve snímku *n+1* ze snímku *n*. Unikátní instance třídy `gm_KalmanFilter` může být inicializovaná dvěma způsoby. První způsob inicializuje instanci třídy `gm_KalmanFilter` pro předvídání nové pozice na základě aktuální pozice, rychlosti a zrychlení, tento způsob je ve třídě `gm_Face` signalizován atributem `m_kf_type` nastaveným na hodnotu 6. Druhý způsob inicializuje instanci třídy `gm_KalmanFilter` pro předvídání nové pozice pouze z aktuální pozice a aktuální rychlosti, tento způsob je ve třídě `gm_Face` signalizován atributem `m_kf_type` nastaveným na hodnotu 4. První způsob inicializace v porovnání s druhým způsobem dosahuje větší přesnosti prováděných předpovědí na úkor nižší rychlosti těchto předpovědi. V realizovaném řešení je však predikovaná pozice sledované tváře dále zpřesňována pomocí cílené detekce (kapitola 5.3) a proto lze používat méně přesný odhad za cenu vyšší rychlosti. Z předchozí věty plyne důvod, proč byl druhý způsob inicializace instance třídy `gm_KalmanFilter` zvolen jako defaultní pro realizovanou knihovnu. Závěrem je ke třídě `gm_KalmanFilter` vhodné poznamenat, že

třída gm_KalmanFilter ve svém principu vychází z třídy KalmanFilter implementované v rámci knihovny OpenCV.

Třída gm_Face je používána jako datový typ pro sledované tváře. Třída gm_Face zahrnuje metody InitKalman, CounterSearchArea, Predict, Draw a funkce ReDetect a FoundInDetectedArray.

Metoda InitKalman vytvoří unikátní instanci třídy gm_KalmanFilter. Odkaz na vytvořenou instanci je poté uložen do atributu m_kf. Na konec opisované metody je zavolána funkce init z vytvořené instance. Funkcí init je provedena inicializace vytvořené instance podle hodnoty atributu m_kf_type, jenž je součástí třídy gm_Face. Je vhodné poznamenat, že atribut m_kf_type byl vysvětlen již dříve v textu a proto zde nebude znovu popisován.

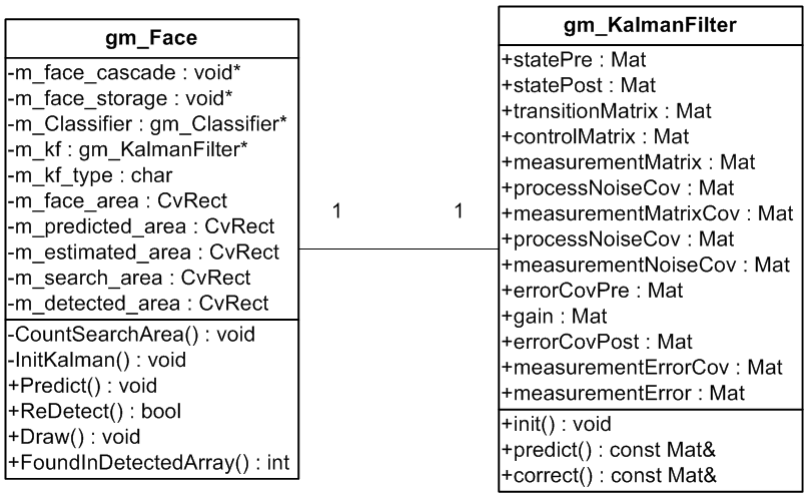
Metoda Predict provádí předpověď pozice sledované tváře ve snímku *n+1* z pozice ve snímku *n*. Pozice ve snímku *n* je získána z atributu m_face_area. Popisovaná funkce funguje na základě přidružené instance třídy gm_KalmanFilter, kdy s každým zavoláním funkce Predict ze třídy gm_Face jsou zavolána funkce predict a corect přidružené instance třídy gm_KalmanFilter. Na základě přidružené funkce predict, resp. corect je naplněn atribut m_predicted_area, resp. m_estimated_area. Závěrem popisované funkce Predict je zavolána metoda CountSearchArea. Zmíněná metoda CountSearchArea z atributů m_predicted_area a m_estimated_area určí obdélník vymezené oblasti pro cílenou detekci, jenž následně uloží do atributu m_search_area.

Funkce ReDetect provádí cílenou detekci ve vymezené oblasti. Vymezená oblast cílené oblasti je dána atributem m_search_area. Výstupní hodnotu cílené detekce je hodnota TRUE v případě nalezení tváře ve vymezené oblasti, resp. hodnota FALSE v opačném případě. V rámci této funkce je volána OpenCV funkce cvHaarDetectObject se vstupním parametrem CV_HAAR_FIND_BIGGEST_OBJECT, viz kapitola 5.2. Výstupní hodnota této OpenCV funkce je uložena do jednoho, nebo dvou atributů patřičné sledované tváře. Počet atributů, do nichž je výstupní hodnota uložena, je dán specifickým vstupním parametrem funkce ReDetect s názvem isControl. Pokud je funkce ReDetect zavolána s hodnotou specifického vstupního parametru FALSE, je výstupní hodnota zmíněné OpenCV funkce uložena pouze do atributu m_face_area. Pokud je funkce ReDetect zavolána s hodnotou specifického vstupního parametru TRUE, pak je hodnota zmíněné OpenCV funkce uložena do atributu m_face_area a navíc i do atributu m_detected_area. Jinými slovy, specifický vstupní parametr isControl nastavený na hodnotu TRUE symbolizuje, že je funkce ReDetect prováděna nad *kontrolním snímkem*, a že má být výstupní hodnota OpenCV funkce uložena i do atributu, jenž zastupuje sledovanou tvář při porovnání sledované tváře s výsledky *kontrolní detekce*.

Funkce FoundInDetectedArray provádí vyhledání sledované tváře v poli nově detekovaných tváří. Toto hledání je prováděno na základě atributu m_detected_area, jenž je plněn

pouze funkcí `ReDetect` prováděnou nad *kontrolním snímkem*. Nad stejným *kontrolním snímkem* je vždy také provedena i *kontrolní detekce* tváří v celém obraze, jejímž výstupem je pole nově detekovaných tváří. Pokud se sledovaná tvář v *kontrolním obraze* vyskytovala, pak se nutně musí vyskytovat i v poli nově detekovaných tváří, resp. pak se v poli nově detekovaných tváří musí vyskytovat tvář odpovídající atributu `m_detected_area`. Výstupní hodnota této funkce je hodnota `TRUE` v případě, že se v poli nově detekovaných tváří vyskytovala tvář odpovídající příslušnému třídnímu atributu `m_detected_area`, resp. hodnota `FALSE` v opačném případě.

Metoda `Draw` provádí vykreslení jedné tváře do výstupního obrazu, konkrétně vykreslení obdélníka daného atributem `m_face_area` do výstupního obrazu OpenCV funkcí `cvRectangle`.



Obrázek 6.2: Zjednodušený třídní diagram sledované tváře

6.2 Praktická implementace ukázkové aplikace

V této kapitole budou uvedeny dvě vzorové aplikace. Tyto vzorové aplikace prezentují dva nejvhodnější způsoby využití realizované knihovny pro detekci obličejů ve videu. První aplikace provádí real-time detekci obličejů ve videu. Druhá aplikace provádí detekci v obraze, jenž je oproti situaci v reálné scéně zpožděn cca o 2 sekundy, přičemž toto zpoždění zůstává po celou dobu běhu konstantní. První aplikace je vhodná např. pro detekci obličeje v obraze získaném z webkamery, kde jakékoliv zpoždění ubírá na přirozenosti. Druhá aplikace nalezne své uplatnění např. při detekování tváří v dříve nahraném videu, nebo také např. v případě dopravních kamer, kdy není důležité, zda bude řidič identifikován okamžitě, nebo se zpožděním 2 sekund.

Obě zmíněné aplikace vycházejí z návrhu uvedeného v kapitole 5.3, kdy druhá aplikace oproti původnímu návrhu navíc přidává buffer pro snímky vstupní videosekvence. Obě řešení budou blíže popsány v následujících dvou podkapitolách.

6.2.1 Ukázková aplikace bez bufferu

Tato aplikace je napsána jako konzolová aplikace. Zdrojový kód této aplikace je stejně jako navržená knihovna napsán v jazyce C++. Zdrojový soubor této aplikace je uložen na přiloženém cd ve složce demo01 pod názvem demo01.cpp. Spustitelná verze této aplikace pro MS Windows je na přiloženém souboru reprezentována soborem domo01.exe, jenž je uložen ve složce demo01\bin. Tato aplikace neočekává žádné vstupní parametry při spuštění.

Popisovaná ukázková aplikace do značné míry vychází návrhu řešení z kapitoly 5.3. Zmíněný návrh z kapitoly 5.3 byl však vytvořen s důrazem na správné pochopení hlavních principů, proto byl popsán drobným zkreslením oproti skutečnému řešení. Obecná funkce ukázkové aplikace se tak od obecné funkce navrženého řešení liší v rozdělení jednotlivých operací do vláken. Ukázková aplikace provádí operace aktualizace a načítání *kontrolního snímku* v hlavním vlákně, zatímco obecná funkce navrženého řešení prováděla tyto operace ve vedlejším vlákně.

Průběh ukázkové aplikace lze rozdělit do 8 fázi. Ještě před samotným popisem průběhu ukázkové aplikace je však vhodné poznamenat, že ukázková aplikace používá jako vstupní video sekvenci výstup z kamery připojené k počítači pomocí USB. Změna na používání vstupního video souboru místo zmíněné kamery by spočívala v editaci jediného řádku zdrojového souboru, konkrétní řádek a konkrétní způsob editace bude uveden při popisu druhé fáze běhu ukázkové aplikace.

V první fázi dochází k inicializaci klasifikátoru pro účely *kontrolní detekce* a inicializaci klasifikátoru pro účely *cílené detekce*. První zmíněná inicializace je realizována funkcí `gm_prepare_MY_FaceDetector`. Druhá zmíněná inicializace je pak realizována funkcí `gm_prepare_CV_FaceDetector`. První fáze dále pokračuje otestováním podpory SIMD instrukci pomocí funkce `SSEAccelerationSupported` a výstup z této funkce je uložen do proměnné `sse_support`. Dále je vytvořeno hlavní okno aplikace pomocí OpenCV funkce `cvNamedWindow`.

Ve druhé fázi je jako zdroj vstupní videosekvence určena připojená kamera pomocí OpenCV funkce `cvCaptureFromCam`. Dále je v této fázi načten první *kontrolní snímek* ze vstupní videosekvence pomocí OpenCV funkce `cvQueryFrame`. Je vhodné poznamenat, že v úvodu zmíněná změna aplikace pro detekování nad vstupním video souborem místo nad vstupní kamerou by spočívala v záměně OpenCV funkce `cvCaputreFromCam` za OpenCV funkci `cvCaputreFromAvi`.

Ve třetí fázi je provedena *kontrolní detekce*. *Kontrolní detekce* je v rámci této aplikaci vždy prováděna nad *kontrolním snímkem* pomocí funkce `gm_MY_FaceDetector` se vstupním parametrem `sse_support` nastaveným dle stejnojmenné proměnné naplněné v první fázi. Dle provedené *kontrolní detekce* je následně naplněno pole detekovaných tváří. Pole detekovaných tváří je typu `std::vector<CvRect>`. Podle pole detekovaných tváří je dále naplněno pole sledovaných tváří prostřednictvím funkce `AddTrackFaces`. Pole sledovaných tváří je typu `std::vector<gm_Face>`.

Čtvrtá fáze začíná vytvořením vedlejšího vlákna. Ve vedlejším vlákně po celou dobu běhu ukázkové aplikace dochází pouze k jediné operaci a to ke *kontrolní detekci* nad *kontrolním snímkem*.

Přičemž je vhodné poznamenat, že načítání *kontrolního snímku* a zpracování výstupu *kontrolní detekce* je prováděno v rámci hlavního vlákna.

Pátá fáze začíná testem, zda ve vedlejším vlákně stále probíhá *kontrolní detekce*. Pokud *kontrolní detekce* ve vedlejším vlákně stále probíhá, pak program v hlavním vlákně pokračuje na začátek sedmé fáze. V případě, že *kontrolní detekce* ve vedlejším vlákně již neprobíhá, je v hlavním vlákně smazáno pole detekovaných tváří a následně jsou do tohoto pole přidány všechny tváře detekované v rámci právě dokončené *kontrolní detekce*. Na konci páté fáze je načten nový *kontrolní snímek* ze vstupní videosekvence prostřednictvím OpenCV funkce `cvQueryFrame`. Zmíněný *kontrolní snímek* je poté předán vedlejšímu vláknu, kde je nad ním spuštěna nová *kontrolní detekce*.

V šesté fázi je nejprve porovnáno pole sledovaných tváří s polem nově detekovaných tváří prostřednictvím funkce `gm_CompareFaces`. Následně je pak pro každou položku, jež zůstala v poli detekovaných tváří, přidán odpovídající záznam do pole sledovaných tváří prostřednictvím funkce `AddTrackFaces`. Dále je v této fázi pro každou tvář z pole sledovaných tváří zavolána její třídní funkce `Predict` a `ReDetect`, přičemž funkce `ReDetect` je volána se specifickým vstupním parametrem `isControl` nastaveným na hodnotu `TRUE`. Dále je v této fázi pro každou položku pole sledovaných tváří zavolána její třídní funkce `Draw`, jež zakreslí příslušnou tvář do *kontrolního snímku*. Na konec šesté fáze je pak *kontrolní snímek* ještě zobrazen OpenCV funkcí `cvShowImage`.

Sedmá fáze začíná načtením vstupního obrazu ze vstupní videosekvence, resp. z připojené kamery. Toto načtení je provedeno opět prostřednictvím OpenCV funkce `cvQueryFrame`. Tato fáze dále pokračuje zavoláním třídních funkcí `Predict` a `Redetect` pro každou položku z pole sledovaných tváří., přičemž funkce `ReDetect` je volána se specifickým vstupním parametrem `isControl` nastaveným na hodnotu `FALSE`. Dále je poté pro každou položku pole sledovaných tváří zavolána její třídní funkce `Draw`, jež zakreslí příslušnou tvář do vstupního snímku. Na konec sedmé fáze je pak *kontrolní snímek* ještě zobrazen prostřednictvím OpenCV funkce `cvShowImage`.

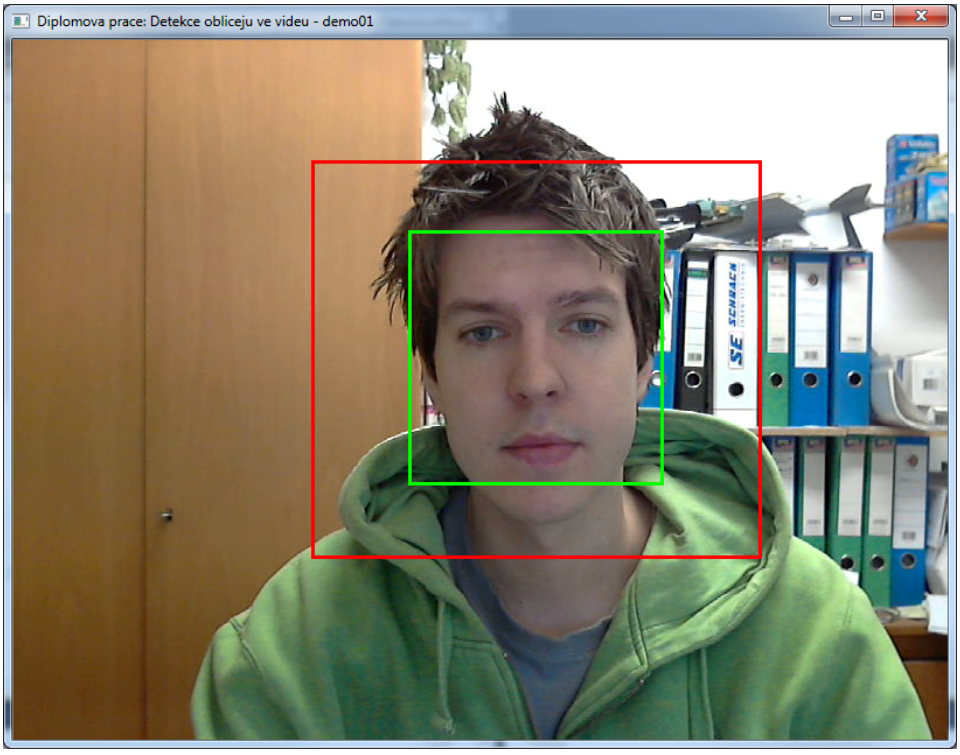
V Osmé fázi je vyhodnoceno, zda již byla v průběhu činnosti ukázkové aplikace stisknuta klávesa pro konec programu. Mezi klávesy ukončující program patří klávesa ‘q‘ a ESC. Stisk klávesy je konkrétně detekován pomocí OpenCV funkce `cvWaitKey`. Pokud byla stisknuta klávesa pro ukončení programu, pak program v hlavním vlákně přechází do deváté fáze. V opačném případě program v hlavním vlákně pokračuje na začátek páté fáze.

V deváté fázi nejprve dochází na dokončení *kontrolní detekce* ve vedlejším vlákně. Bezprostředně po dokončení *kontrolní detekce* je korektně ukončeno vedlejší vlákno. V deváté fázi je pak dále provedeno uvolnění vstupní videosekvence pomocí OpenCV funkce `cvReleaseCapture`, poté uvolnění klasifikátoru pro cílenou detekci pomocí OpenCV funkcí `cvReleaseMemStorage` a `cvReleaseHaarClassifierCascade`. Na závěr deváté fáze je ještě ukončeno i hlavní okno programu pomocí OpenCV funkce `cvDestroyWindow` a následně končí i celá ukázková aplikace.

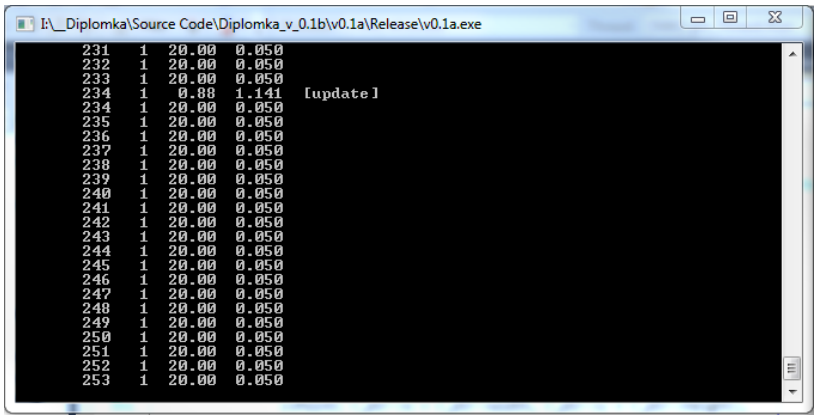
Výhodou této ukázkové aplikace je real-time detekce tváří v obraze na základě vysoce kvalitního klasifikátoru, jenž dosahuje skvělé úspěšnosti při detekcích tváří v obraze (výsledky klasifikátoru viz kapitola 7.1). Tato ukázková aplikace by dokázala běžet s rychlostí přes **52 fps** (počet snímků za sekundu – z anglického ***f**rame **p**er **s**econd*) pro scénu s jednou sledovanou tváří při rozlišení vstupní videosekvence 800x600 pixelů. Rychlost kolem 50 fps je však pro většinu dnešních kamer nedosažitelná, proto je běh této aplikace uměle zpomalen na rychlost 20 fps (perioda 50 ms).

Nevýhodou této aplikace však je její závislost na počtu tváří, kdy od počtu sledovaných tváří, viz testování v kapitole 7.2.2. Přesto však tato aplikace dosahuje hranice cca 14 fps pro vstupní obrazy s 10 tvářemi pořízenými v průměrném rozlišení. Další nevýhodou této aplikace je přibližně sekundová prodleva při detekci nové tváře, způsobená časem potřebným k dokončení *kontrolní detekce*. Ideálním prostorem pro nasazení této aplikace je například detekce tváří během video přenosů prostřednictvím komunikátoru *skype*.

Na obrázku 6.1 je zobrazeno hlavní okno ukázkové aplikace. Červený rámeček v obrázku 6.1 prezentuje predikovanou vymezenou oblast, v níž byla provedena cílená detekce. Výsledek cílené detekce je pak symbolizován zeleným rámečkem a odpovídá nalezené tváři v obraze. Je vhodné poznamenat, že červený rámeček je na obrázku 6.1 zobrazen pouze pro demonstrační účely této textové práce, ve finální verzi ukázkové aplikace tento rámeček zobrazován není. Na obrázku 6.2 je zobrazeno konsolové okno této ukázkové aplikace, jež v průběhu detekce vypisuje kontrolní výpisy informující o pořadovém čísle vstupního snímku, o počtu sledovaných/detekovaných tváří, o rychlosti sledování/detekce, o dokončení *kontrolní detekce* (ukončené řetězcem [update]).



Obr. 6.1: Hlavní okno ukázkové aplikace (shodné pro obě ukázkové aplikace)



Obr. 6.2: Konsolový výstup ukázkové aplikace (principielně shodné pro obě ukázkové aplikace)

6.2.2 Ukázková aplikace s bufferem

Tato aplikace je stejně jako předchozí ukázka napsána jako konzolová aplikace v jazyce C++. Soubor se zdrojovým kódem této ukázky je uložen na přiloženém cd ve složce demo02 pod názvem demo02.cpp. Spustitelná verze popisované ukázky pro operační systém MS Windows XP a vyšší je uložena rovněž na přiloženém cd ve složce demo02\bin pod názvem demo02.exe. Tato aplikace, stejně jako předchozí, neočekává při spuštění žádné přídavné parametry.

Popisovaná aplikace vychází z návrhu řešení, resp. ze zjednodušeného blokového schéma uvedeného v kapitole 5.4. Princip převedení návrhu 5.3 s využitím funkcí z realizované knihovny na ukázkovou aplikaci bez bufferu byl popsán v rámci kapitoly 6.2.1. Princip převedení návrhu 5.4 pomocí realizované knihovny na ukázkovou aplikaci s bufferem nepřináší nic nového, v porovnání s principem uvedeným v kapitole 6.2.1, a proto zde nebude tento princip znovu popisován. Je pouze vhodné zmínit, že jako buffer byl implementován jako `std::vector<IplImage*>`.

Výhodou této ukázkové aplikace je snadná implementace bez zbytečného porovnávání sledovaných a detekovaných tváří, což se značí zejména nevyužitím funkce `gm_CompareFaces` z realizované knihovny. Další výhodou této aplikace je stejně jako v případě první ukázkové aplikace rychlost činnosti programu, kdy tato ukázková aplikace detekuje, resp. sleduje tváře ve vstupní video sekvenci se stejnou rychlostí jako předchozí ukázka bez bufferu.

Nevýhodou této ukázkové aplikace je opět závislost na počtu sledovaných, resp. detekovaných tváří, viz testy v kapitole 7.2.2. Další nevýhodou této aplikace je *n* sekundové zpoždění zobrazovaných obrazů oproti vstupní video sekvenci (viz návrh v kapitole 5.4).

Ideálním prostorem pro nasazení této aplikace je například detekce tváře řidiče v obrazech z dopravní kamery, dále pak detekce tváří v dříve nahraných video souborech atd. Nevhodným prostorem pro nasazení této aplikace je například detekce obličeje z webkamery, kdy *n* sekundová prodleva může být pro uživatele dosti matoucí.

7 Testování realizované knihovny

Tato kapitola popisuje průběh testování realizované knihovny a poté vyhodnocuje výsledky z těchto testů. Testování probíhalo na PC s procesorem Intel Core 2 Duo s modelovým značením E8400, jenž byl taktován na 3 GHz. Zmíněný počítač byl dále vybaven 8 GB operační pamětí, jež byla tvořena čtyřmi 2 GB moduly značky A-DATA s časováním CL4 (4-4-4-12). Je vhodné poznamenat, že jakákoliv testování realizované karty jsou nezávislé na grafické kartě, jelikož dle zadání diplomové práce realizovaná knihovna nesmí využívat žádné akcelerace prostřednictvím grafické karty.

Tato kapitola je dále rozčleněna do 4 podkapitol, kdy v první podkapitole jsou uvedeny testy OpenCV klasifikátorů, ve druhé podkapitole jsou uvedeny testy realizovaných detektorů tváří v obraze, ve třetí podkapitole jsou uvedeny testy procesu souběžného sledování a detekování tváří v obraze a konečné ve čtvrté kapitole je uvedeno vyhodnocení celého testování.

7.1 Testování OpenCV klasifikátorů

V této kapitole jsou uvedeny výsledky testů dvou xml klasifikátorů distribuovaných s OpenCV knihovnou. Zmíněné klasifikátory byly aplikovány prostřednictvím funkce `cvHaarDetectObjects` k detekci tváří v sedmi různých obrazech. První obraz neobsahoval žádnou lidskou tvář, druhý obsahoval jednu tváří, třetí obsahoval 2 tváře, čtvrtý 5 tváří, pátý 10 tváří, šestý 170 tváří a konečně sedmý obsahoval 360 tváří. Všechny obrazy byly pořízeny ve stejném rozlišení 800x600 pixelů. Výsledky detekcí pomocí obou zmíněných klasifikátorů jsou uvedeny v tabulce 7.1, kde jsou konkrétně uvedeny výsledky dosažené z hlediska úspěšných označení tváří, z hlediska neúspěšných označení tváří a z hlediska rychlosti, s jakou byly detektory schopny v obrazech detekci provést. Je vhodné poznamenat, že za úspěšnou detekci byl považován stav, kdy detektor označil místo v obraze, jež odpovídalo lidské tváři. Naopak za neúspěšnou detekci byl považován stav, kdy detektor označil místo, v němž se lidská tvář nevyskytovala.

Z testů plyne, že detektor založený na klasifikátoru `haarcascade_frontalface_alt.xml` dosahuje vyšší úspěšnosti správné detekce, za cenu nižší rychlosti detekce v porovnání s detektorem používajícím `haarcascade_frontalface_default.xml`. V této diplomové práci byl kladen důraz zejména na úspěšnost detekce, proto realizovaná knihovna využívá klasifikátor s koncovkou *alt*.

Počet tváří v obraze	**0**	**1**	**2**	**5**	**10**	**170**	**360**
Správné detekce (default)	0	1	2	5	**9**	170	**346**
Správné detekce (alt)	0	1	2	5	10	170	352
Nesprávné detekce (default)	0	0	0	1	2	0	0
Nesprávné detekce (alt)	0	0	0	0	0	1	1
Čas detekce (default) [ms]	821	910	937	888	921	2833	2967
Čas detekce (alt) [ms]	1277	1279	1282	1299	1326	3077	3185

Tabulka 7.1: Testování klasifikátorů *frontalface_default* a *frontalface_alt*.

7.2 Testování realizované knihovny

Tato kapitola prezentuje výsledky z testování celé knihovny. Na začátek je v tabulce 7.2 uveden souhrn výsledků z testu rychlosti všech navržených řešení popsaných v kapitole 5. Poté následuje rozbor a zhodnocení zmíněné tabulky. Na konec této podkapitoly jsou pak uvedeny podrobnější testy jednotlivých částí navržené knihovny, konkrétně podrobnější testy optimalizovaného detektoru (podkapitola 7.2.1) a podrobnější testy obou ukázkových aplikací (podkapitola 7.2.2).

		Počet tváří v obraze						
		0	1	2	5	10	170	360
Čas [ms]	Standardní detektor (5.1.2)	1936	2022	1997	2015	1986	2478	2827
	Standardní detektor s opti. průchodem podoken (5.1.4)	1197	1285	1256	1280	1241	1743	2086
	Detektor s SIMD opti. na úrovni obdélníka (5.1.5.1)	1584	1708	1683	1697	1642	2284	2725
	Optimalizovaný detektor (5.1.6)	905	971	940	974	952	1384	1661
	Sledování tváří na základě pozice *x*, rychlosti *v*, zrychlení *a*	0	18	32	68	84	1919	2257
	Sledování tváří na základě pozice *x* a rychlosti *v*	0	18	30	65	79	1447	1686
	Opti. detektor při souběžném sledování na základě *x*, *v*, *a*	1215	1364	1372	1325	1376	4000	4167
	Sledování na základě *x*, *v*, *a* při souběžném opti. detektoru	0	20	35	84	117	2591	2725
	Opti. detektor při souběžném sledování na základě *x* a *v*	1215	1364	1372	1325	1376	4000	4167
	Sledování na základě *x*, a *v* při souběžném opti. detektoru	0	19	33	80	94	1980	1992

Tabulka 7.2: Testování rychlosti (času zpracování jednoho snímku) všech navržených řešení.

Tabulka 7.2 uvádí výsledky testu rychlosti, jež probíhal nad obrazy s 1, 2, 5, 10 a 170 tvářemi. Zmíněné obrazy byly pořízeny ve stejném rozlišení 800x600 pixelů. Tento test proběhl pro všechna navržená řešení popsaná v rámci kapitoly 5.

Ještě před vyhodnocením výsledků z tabulky 7.2 je vhodné znovu připomenout, že všechny detektory prezentované tabulkou 7.2 dosahují naprosto stejnou úspěšnost jako OpenCV detektor používající klasifikátor s koncovkou *alt*. Z tabulky 7.2 pak plyne, že optimalizovaný detektor dosahuje nejvyšší rychlosti detekce pří zachování stejné míry úspěšnosti detekce, jakou dosahoval OpenCV detektor s koncovkou *alt* v tabulce 7.1.

Z tabulky 7.2 dále plyne, že sledování, jež predikuje vymezené oblasti pouze na základě pozice a rychlosti, je nepatrně rychlejší, než sledováním, jež zmíněnou predikci upřesňuje pomocí zrychlení sledované tváře.

Další závěr, jenž lze na základě tabulky 7.2 konstatovat, je, že při souběžném sledování a detekování tváří dochází ke snížení rychlosti optimalizované detekce i sledování tváří. Rychlost detekce však ani za těchto podmínek neklesne pod rychlost dosažnou OpenCV funkcí při stejném klasifikátoru. Rychlost sledování při souběžném detekování pro obrazy se 170 a 360 tvářemi ale klesá pod rychlost, při jaké běží OpenCV detektor samostatně. Z předchozí věty plyne, že pro 170 a 360 tváří začíná být návrh z kapitoly 5.3, resp. 5.4 méně výhodný v porovnání se samostatně běžícím optimalizovaným detektorem tváří. Hranice mezi výhodností a nevýhodností souběžného sledování a detekování oproti samostatně běžícímu detektoru je vysoce závislá na vyhodnocovaném obrazu, resp. na počtu tváří ve vyhodnocovaném obraze. Z předchozí věty plyne, že zmíněnou hranici není možné zvolit defaultně pro všechny scény. Experimentálně však bylo zjištěno, že tato hranice pro jakoukoliv scénu neklesne pod 75 tváří, resp. že pro jakoukoliv scénu, s méně než 75 tvářemi je souběžné sledování a detekování vždy rychlejší, v porovnání se samostatnou detekcí. Tato práce počítá s nasazením ve scénách, v nichž se bude vyskytovat měně, než 75 tváří, proto není scéna testována na počet tváří a dále vyhodnocována, zda se má program přepnout do módu samotného optimalizovaného detekování nebo i nadále zůstat v módu souběžného sledování a detekování.

7.2.1 Testování optimalizovaného detektoru

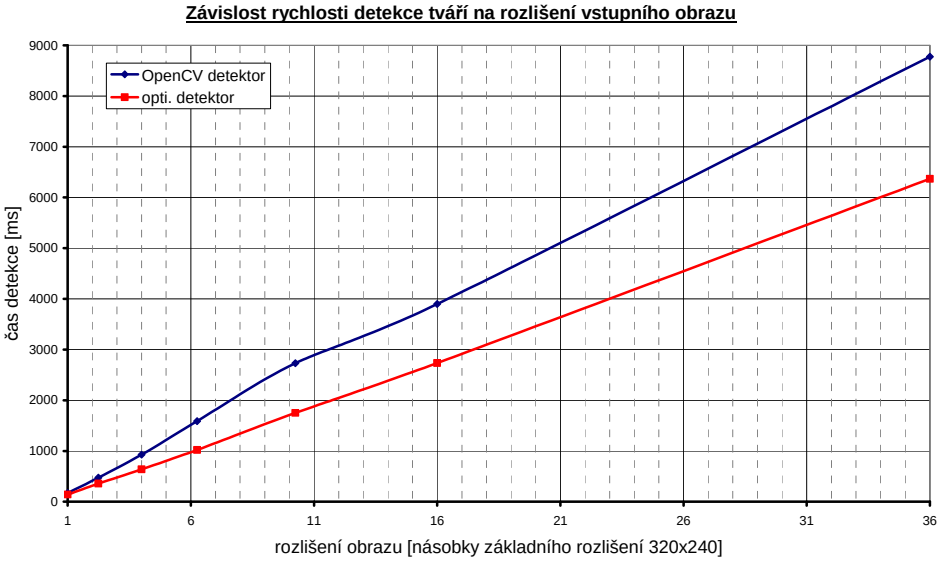
Tato kapitola graficky prezentuje výsledky testování optimalizovaného a OpenCV detektoru, přičemž oba detektory byly použity s klasifikátorem s koncovkou *alt*. Oba detektory byly otestovány třemi sadami testů, viz podkapitoly 7.2.1.1, 7.2.1.2 a 7.2.1.3. První sadou testů byla otestována rychlost obou zmíněných detektoru v závislosti na rozlišení vstupního obrazu a také rychlost v závislosti na počtu tváří ve vstupním obraze. Druhou sadou testů byla otestována rychlost v závislosti na kroku procházení podoken v obrazech pořízených s odlišným rozlišením a konstantním počtem tváří. Třetí sada testů byla použita k otestování rychlosti a úspěšnosti optimalizovaném detektoru v závislosti na kroku procházení podoken ve vstupních obrazech s různým počtem tváří a konstantním rozlišením.

7.2.1.1 První sada testů optimalizovaného detektoru

První sada testů obou detektorů proběhla na sedmi šesticích obrazů. Kdy každá šestice byla tvořena obrazy s 0, 1, 2, 5, 10 a 170 tvářemi. První šestice obrazů pak byla pořízena v rozlišení 320x240, druhá v rozlišení 480x360, třetí v rozlišení 640x480, čtvrtá v rozlišení 800x600, pátá v rozlišení 1024x768, šestá v rozlišení 1280x960 a sedmá šestice byla pořízena v rozlišení 1920x1440. Kompletní výsledky z první sady testů jsou uvedeny v tabulce P-1 v příloze 3.

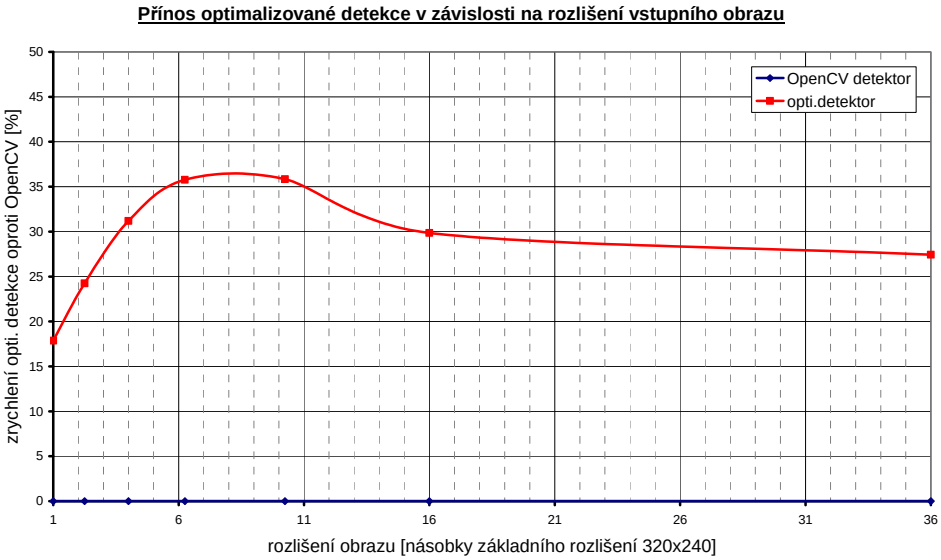
Z první sady testů vzešlo 6 křivek, jež znázorňují rychlosti OpenCV detekce tváří v obrazech s 0, 1, 2, 5, 10 a 170 tvářemi v závislosti na rozlišení těchto obrazů. Dále bylo z první sady testů vytvořeno 6 křivek, jež analogicky znázorňují rychlosti optimalizovaného detektoru tváří nad obrazy s 0, 1, 2, 5, 10 a 170 tvářemi v závislosti na rozlišení těchto obrazů. Těchto šest křivek je graficky znázorněno v příloze 2 na obrázku P-2. Poté byly průměrem první šestice, resp. druhé šestice křivek

vytvořeny dvě průměrové křivky, jež znázorňují rychlost OpenCV detektoru tváří a rychlost optimalizovaného detektoru tváří nad obrazy s průměrným počtem tváří v závislosti na rozlišení těchto obrazů. Obě zmíněné křivky jsou graficky znázorněny na obrázku 7.1.



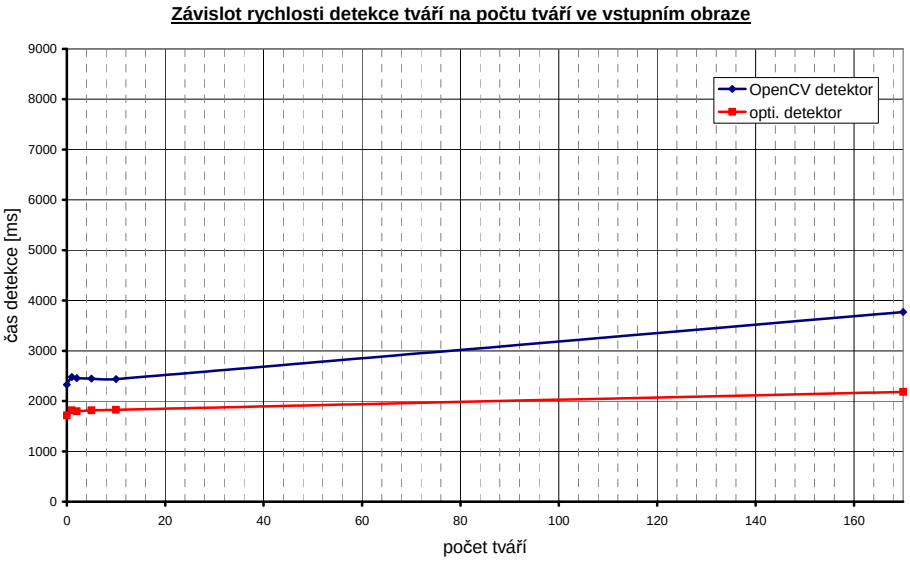
Obrázek 7.1: Grafické znázornění závislosti rychlosti detekce tváří na rozlišení vstupního obrazu.

Dále byla vytvořena třetí šestice křivek, viz obrázek P-3 v příloze 2, jež prezentuje procentuální zrychlení optimalizované detekce oproti OpenCV detekci v obrazech s 0, 1, 2, 5, 10 a 170 tvářemi v závislosti na rozlišení těchto obrazů. Třetí šestice vznikla vydělením šesti rozdílových křivek šesti již dříve zmíněnými křivkami, jež znázorňují rychlosti OpenCV detektoru v závislosti na rozlišení obrazu s 0, 1, 2, 5, 10 a 170 tvářemi. Šestice rozdílových křivek pro změnu vznikla odečtením šesti křivek, jež znázorňují rychlosti OpenCV detekce tváří v obrazech s 0, 1, 2, 5, 10 a 170 tvářemi v závislosti na rozlišení těchto obrazů od šesti křivek, které znázorňují šest analogických závislostí pro optimalizovaný detektor tváří. Zprůměrováním šestice z obrázku P-3 v příloze 2 byla dále vytvořena křivka, jež znázorňuje urychlení optimalizované detekce tváří oproti OpenCV detekci tváří nad obrazy s průměrným počtem tváří v závislosti na rozlišení těchto obrazů, viz obrázek 7.2.



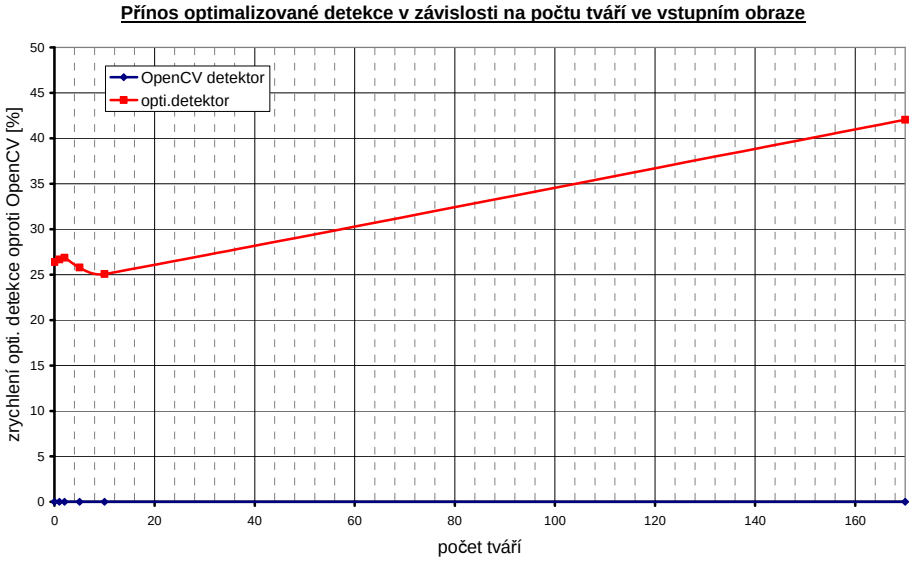
Obrázek 7.2: Graf závislosti přínosu optimalizované detekce tváří na rozlišení vstupního obrazu.

Z první sady testů dále také vzešlo 7 křivek, jež znázorňují rychlosti OpenCV detekce tváří v obrazech pořízených v 7 různých rozlišeních v závislosti na počtu tváří v těchto obrazech. Poté bylo ze zmíněné sady testů vytvořeno 7 křivek, jež analogicky znázorňují rychlosti optimalizované detekce tváří nad obrazy pořízenými v 7 různých rozlišeních v závislosti na počtu tváří v těchto obrazech. Druhá zmíněná sedmice křivek je graficky zobrazena v souhrnném grafu na obrázku P-4 v příloze 2. Z těchto dvou sedmic opět vzešly dvě průměrové křivky, kdy jedna, resp. druhá znázorňuje rychlost OpenCV detektoru tváří, resp. rychlost optimalizovaného detektoru tváří v obrazech s průměrným rozlišením v závislosti na počtu tváří v těchto obrazech Obě křivky jsou zobrazeny na obrázku 7.3.



Obrázek 7.3: Grafické znázornění závislosti rychlosti detekce tváří na počtu tváří

Ze stejné sady testů bylo vytvořeno ještě dalších 7 křivek, viz obrázek P-5 v příloze 2, jež reflektují urychlení optimalizované detekce oproti OpenCV detekci nad obrazy se 7 různými rozlišeními v závislosti na počtu tváří. Těchto 7 křivek vzniklo analogickým postupem jako šestice z obrázku P-3 v příloze 2. Průměrem sedmi zmíněných křivek nakonec vznikla ještě poslední křivka, jež znázorňuje urychlení optimalizované detekce tváří oproti OpenCV detekci tváří v obrazech s průměrným rozlišením v závislosti na počtu tváří v těchto obrazech. Zmíněná průměrová křivka je znázorněna na následujícím obrázku 7.4.

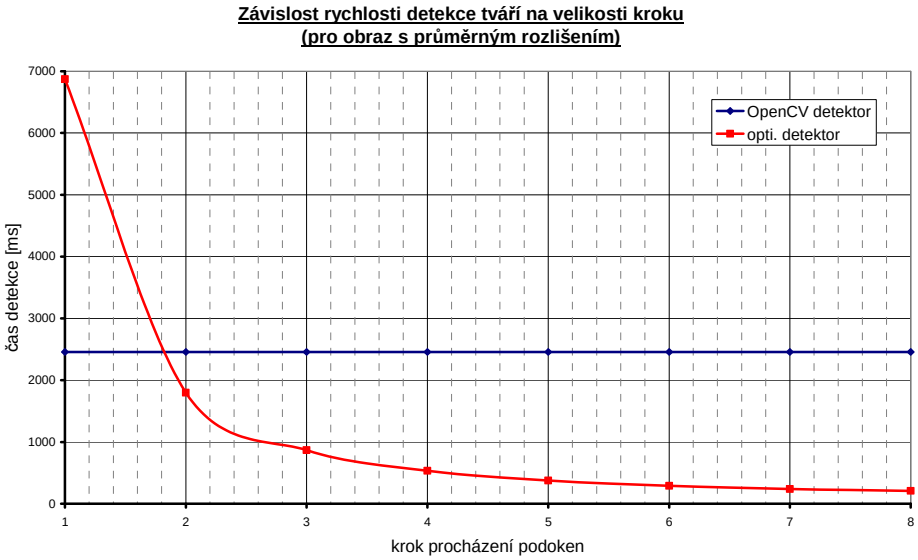


Obrázek 7.4: Graf závislosti přínosu optimalizované detekce tváří na počtu tváří ve vstupním obraze.

7.2.1.2 Druhá sada testů optimalizovaného detektoru

Druhá sada testů optimalizovaného a OpenCV detektoru proběhla na sedmi obrazech se dvěma tvářemi. První obraz byl v rozlišení 320x240, druhý v rozlišení 480x360, třetí v rozlišení 640x480, čtvrtý v rozlišení 800x600, pátý v rozlišení 1024x768, šestý v rozlišení 1280x960 a sedmý obraz byl pořízen v rozlišení 1920x1440. Nad těmito sedmi obrazy byl postupně spouštěn detektor s 8 různými volbami kroku procházení podoken, kdy velikost kroku byla v rozmezí od 1 do 8. Kompletní výsledky z druhé sady testů jsou uvedeny v tabulce P-2 v příloze 3.

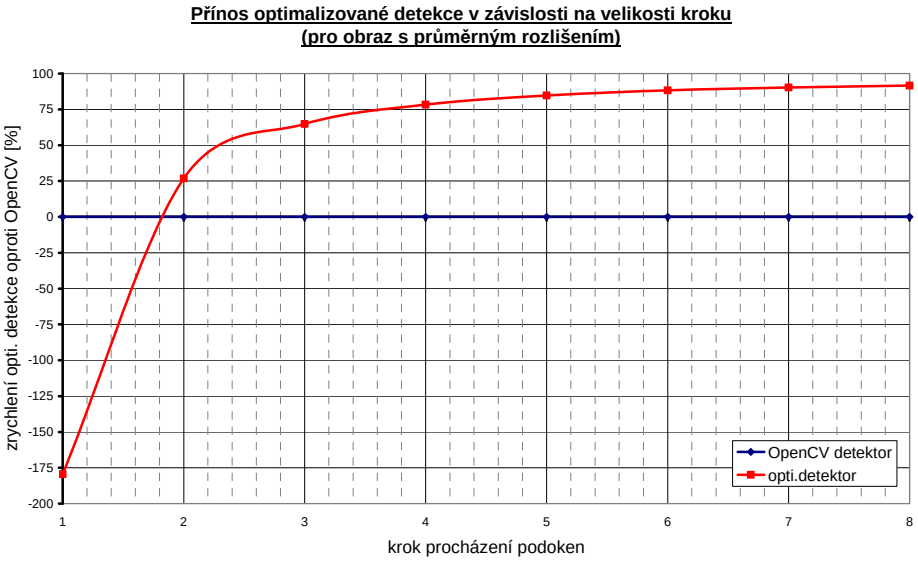
Z druhé sady testů nejprve vzešla sedmice křivek, jež znázorňují rychlosti optimalizovaného detekce tváří v obrazech se dvěma tvářemi pořízených v 7 různých rozlišeních v závislosti na kroku procházení podoken. Zmíněná sedmice křivek je znázorněna na obrázku P-6 v příloze 2. Zmíněných sedm křivek bylo dále zprůměrováno, čímž vznikla nová křivka znázorňující rychlost optimalizované detekce tváří v obraze s průměrným rozlišením a dvěma tvářemi v závislosti na kroku procházení podoken. Tato průměrová křivka je vykreslena na obrázku 7.5. V obrázku 7.5 je pro srovnání uvedena i referenční rychlost OpenCV detektoru pro tentýž obraz s průměrným rozlišením a dvěma tvářemi.



Obrázek 7.5: Graf závislosti rychlosti detekce tváří na kroku pro obraz s průměrným rozlišením.

Dále bylo z druhé sady testu vytvořeno nových 7 křivek, jež reflektují urychlení optimalizovaného detektoru tváří oproti OpenCV detektoru tváří v závislosti na kroku procházení podoken v obrazech se 7 různými rozlišeními, viz obrázek P-7 v příloze 2. Těchto 7 křivek vzniklo analogickým postupem jako šest křivek z obrázku P-3, konkrétní postup viz kapitola 7.2.1.1.

Závěrem byla z druhé sady testů ještě vytvořena křivka znázorňující urychlení optimalizované detekce tváří oproti OpenCV detekci tváří v obraze s průměrným rozlišením a dvěma tvářemi v závislosti na kroku procházení podoken, viz obrázek 7.6. Tato křivka vznikla průměrem sedmi křivek z obrázku P-7 v příloze 2.

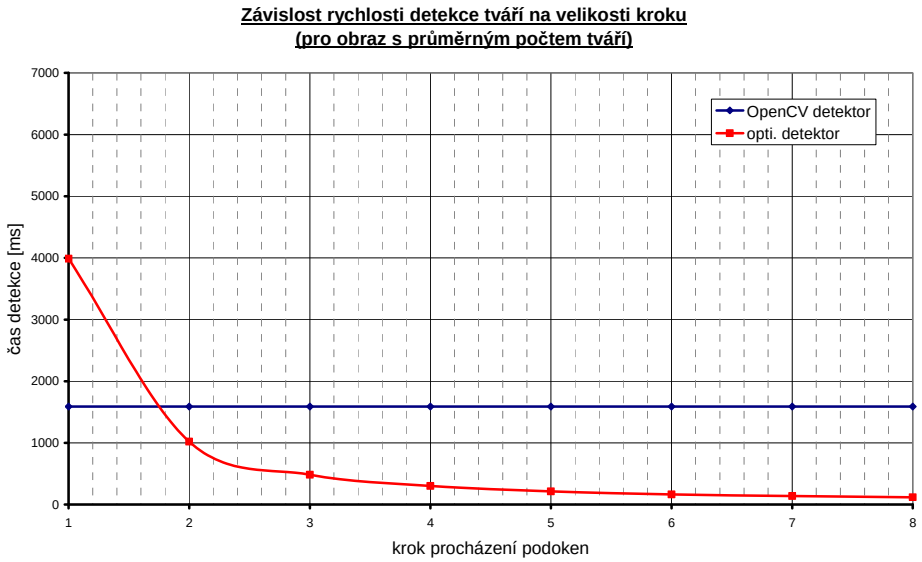


Obrázek 7.6: Graf závislosti přínosu optimalizované detekce na kroku (obraz s průměrným rozlišením).

7.2.1.3 Třetí sada testů optimalizovaného detektoru

Třetí sada testů optimalizovaného a OpenCV detektoru proběhla nad šesti obrazy s konstantním rozlišením 800x600 pixelů. První obraz neobsahoval žádnou tvář, druhý obsahoval 1 tvář, třetí 2 tváře, čtvrtý 5 tváří, pátý 10 tváří a šestý obraz obsahoval 170 tváři. Nad těmito šesti obrazy bylo provedeno 8 skupin testů optimalizovaného detektoru, kdy jednotlivé skupiny se od sebe lišily volbou kroku procházení podoken. Velikost kroku se při těchto sadách testů pohybovala v rozmezí od 1 do 8. Kompletní výsledky z první sady testů jsou uvedeny v tabulkách P-3 a P-4 v příloze 3.

Ze třetí sady testů nejprve vzešla šestice křivek, jež reprezentuje závislost rychlosti optimalizovaného detektoru na kroku procházení podoken pro 6 obrazů s různým počtem tváří pořízených v konstantním rozlišení 800x600 pixelů. Zmíněná šestice křivek je znázorněna na obrázku P-8 v příloze 2. Zmíněných šest křivek bylo dále zprůměrováno, čímž vznikla nová křivka znázorňující závislost rychlosti optimalizovaného detektoru na kroku procházení podoken v obraze s průměrným počtem tváří a konstantním rozlišením 800x600 pixelů. Tato zprůměrovaná křivka je vykreslena na obrázku 7.7. V obrázku 7.7 je pro srovnání uvedena i referenční rychlost OpenCV detektoru v obraze s průměrným počtem tváří pořízeném v rozlišení 800x600 pixelů.

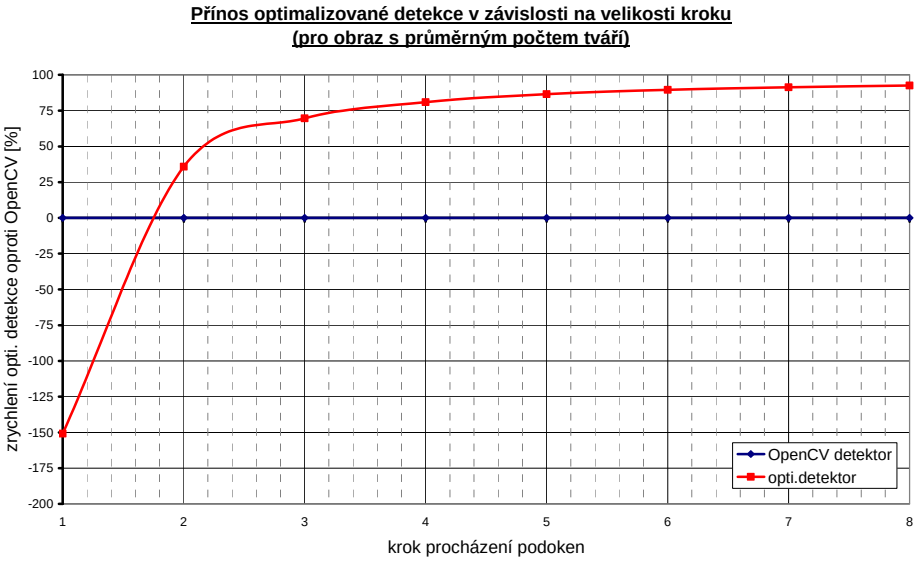


Obrázek 7.7: Graf závislosti rychlosti detekce tváří na kroku pro obraz s průměrným počtem tváří.

Dále bylo z třetí sady testu vytvořeno nových 6 křivek, jež reflektují urychlení optimalizovaného detektoru oproti OpenCV detektoru v závislosti na kroku procházení podoken pro obrazy s konstantním rozlišením 800x600 pixelů obsahující 6 různých počtů tváří., viz obrázek P-9 v příloze 2. Těchto 6 křivek vzniklo analogickým postupem jako šest křivek z obrázku P-3 v příloze 2, podrobný postup viz kapitola 7.2.1.1.

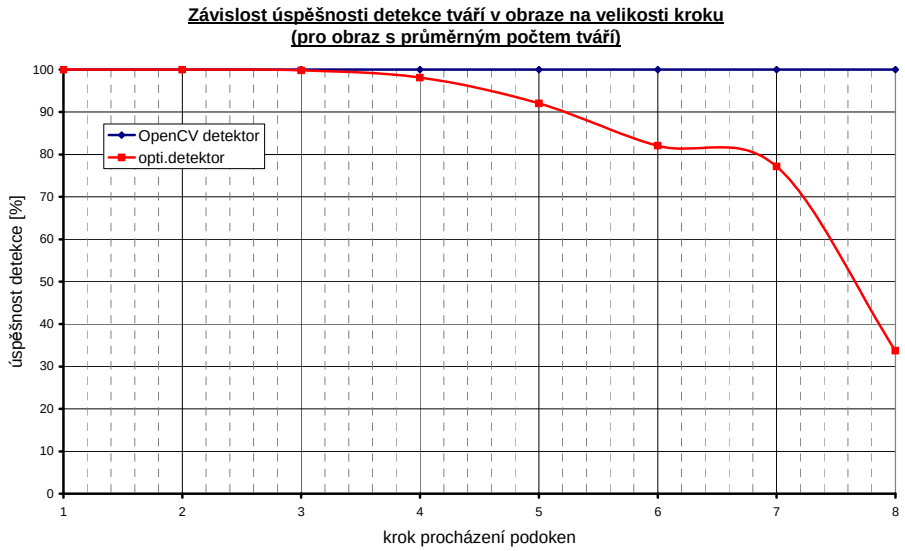
Ze třetí sady byla poté také vytvořena křivka znázorňující urychlení optimalizovaného detektoru oproti OpenCV v závislosti na kroku procházení podoken v obraze s průměrným počtem

tváří a konstantním rozlišením 800x600 pixelů, viz obrázek 7.8. Zmíněná křivka vznikla průměrem sedmi křivek z obrázku 8 v příloze 2.



Obrázek 7.8: Graf závislosti přínosu optimalizované detekce na kroku (průměrný počet tváří v obraze).

V případě třetí sady testů bylo vhodné zaměřit se také na úspěšnost detektoru, jež se rostoucím krokem oproti OpenCV detektoru snižuje, jelikož s rostoucím krokem je vyhodnocováno stále méně podoken. V rámci třetí sady testů tak dále vzniklo šest křivek udávajících průběh úspěšnosti detektoru v závislosti na rostoucím kroku pro šest různých počtů tváří v obrazech s konstantním rozlišením 800x600 pixelů. Tato šestice je zobrazena na obrázku P-10 v příloze 2. Dále bylo vhodné vytvořit i křivku udávající úspěšnost detektoru v závislost na rostoucím kroku pro obraz s průměrným počtem tváří a konstantním rozlišením 800x600 pixelů, viz obrázek 7.9. Zmíněná křivka vznikla průměrem šesti křivek z obrázku P-10 v příloze 2. Na obrázku 7.9 je také uvedena referenční křivka OpenCV detektoru v obraze s průměrným počtem tváří a rozlišením 800x600 pixelů



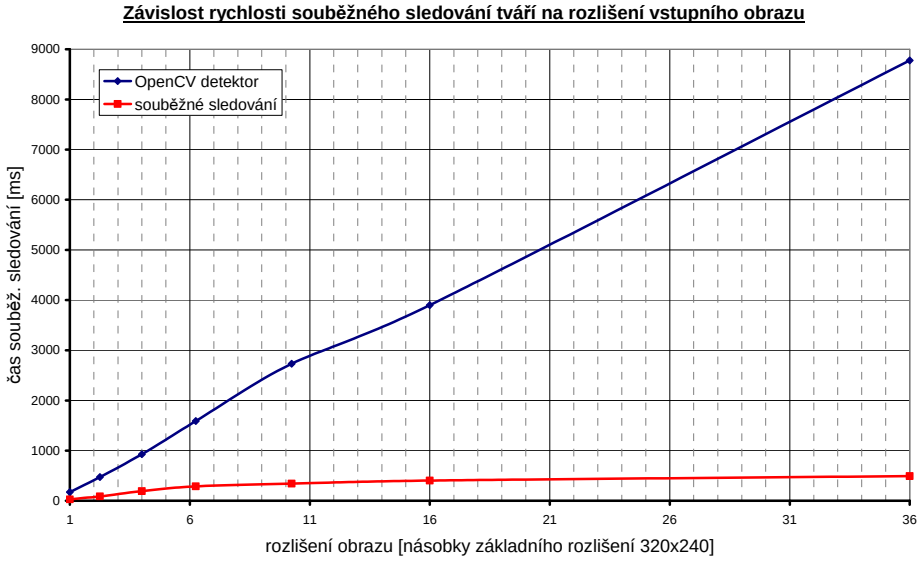
Obrázek 7.9: Graf závislosti úspěšnosti detekce tváří na kroku pro obraz s průměrným počtem tváří.

7.2.2 Testování souběžného sledování a detekování

Tato kapitola graficky prezentuje výsledky ze sady testů provedených pro proces souběžného sledování a detekování tváří a pro OpenCV detektor, přičemž sledování i OpenCV detektor při své činnosti používají klasifikátor s koncovkou *alt*. Touto sadou testů byla otestována rychlost zmíněného sledování tváří i rychlost OpenCV detektoru v závislosti na rozlišení vstupního obrazu a také rychlost v závislosti na počtu tváří ve vstupním obraze. Kompletní výsledky provedených testů nad procesem souběžného sledování a detekování tváří jsou uvedeny v tabulce P-1 v příloze 3.

Zmíněná sada testů proběhla na stejných sedmi šesticích obrazů jako sada testů z podkapitoly 7.2.1.1, přičemž popis těchto sedmi šestic obrazů je uveden v podkapitole 7.2.1.1, a proto zde nebude znovu opakován.

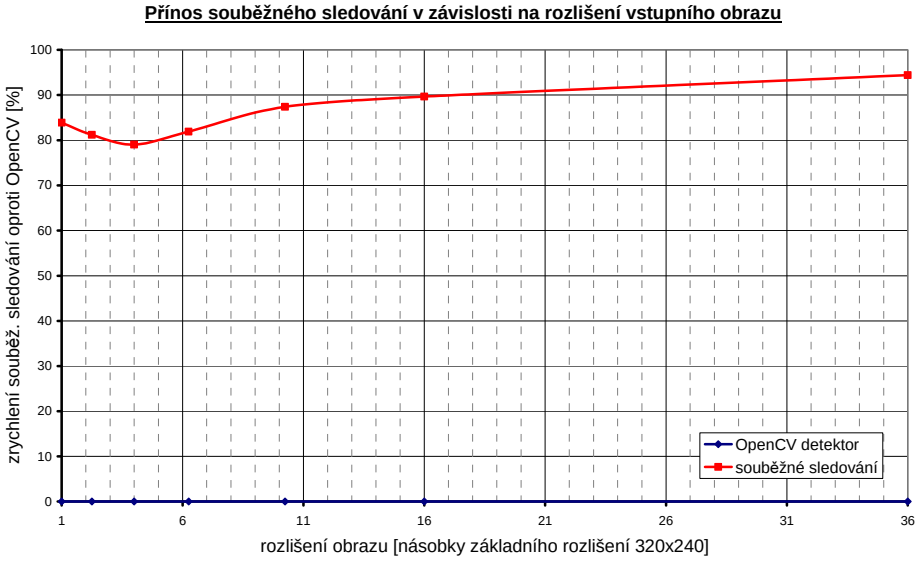
Ze zmíněné sady, stejně jako v kapitole 7.2.1.1, vzešlo 6 křivek, jež znázorňují závislost rychlosti OpenCV detektoru na rozlišení obrazu pro 0, 1, 2, 5, 10 a 170 tváří. Dále pak bylo ze stejné sady testů vytvořeno 6 křivek, jež analogicky znázorňují rychlost procesu souběžného sledování a detekování tváří prováděného nad obrazy s 0, 1, 2, 5, 10 a 170 tvářemi v závislosti na rozlišení obrazu. Těchto šest křivek je graficky znázorněno v příloze 2 na obrázku P-11. Poté byly průměrem první šestice, resp. druhé šestice křivek vytvořeny dvě průměrové křivky, jež znázorňují rychlosti OpenCV detektoru v závislosti na rozlišení obrazu s průměrným počtem tváří, resp. rychlost procesu souběžného sledování a detekování průměrného počtu tváří v závislosti na rozlišení obrazu Obě křivky jsou graficky znázorněny na obrázku 7.10.



Obrázek 7.10: Graf závislosti rychlosti souběžného sledování na rozlišení vstupního obrazu.

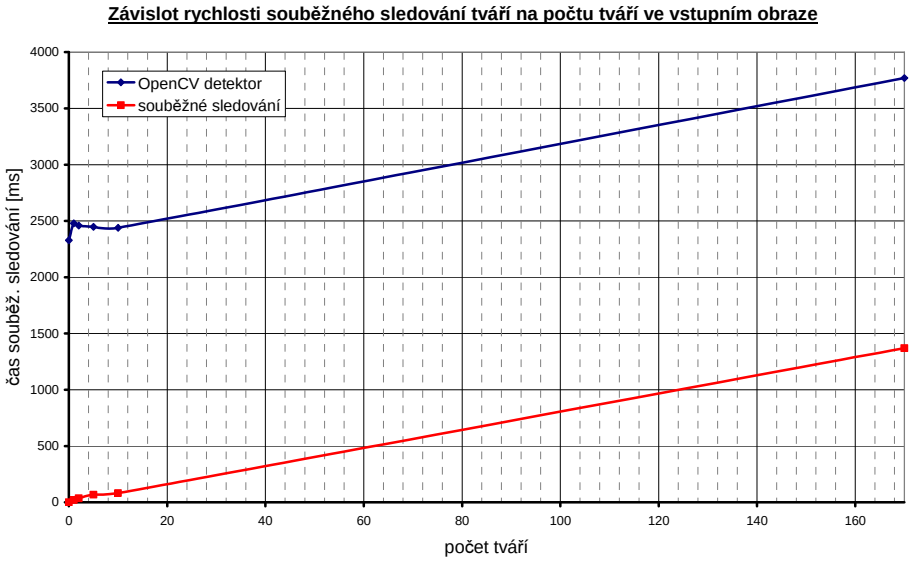
Dále bylo ze zmíněné sady testu vytvořeno nových 6 křivek, viz obrázek P-12 v příloze 2, jež znázorňují míru zrychlení procesu souběžného sledování a detekovaní tváří běžícího nad obrazy s 0, 1, 2, 5, 10 a 170 tvářemi oproti OpenCV detekci tváří v obrazech s 0, 1, 2, 5, 10 a 170 tvářemi v závislosti na rozlišení obrazu. Těchto 6 křivek vzniklo analogicky ke vzniku šesti křivek z obrázku

P-3 v příloze 2. Průměrem této šestice poté vznikla další křivka, viz obrázek 7.11, jež znázorňuje zrychlení procesu souběžného sledování a detekování průměrného počtu tváří oproti OpenCV detekci tváří v obrazech s průměrným počtem tváří v závislosti na rozlišení těchto obrazů.



Obrázek 7.11: Graf závislosti přínosu souběžného sledování na rozlišení vstupního obrazu.

Ze zmíněné sady testů dále také vzešlo 7 křivek, jež znázorňují rychlosti OpenCV detekce tváří nad obrazy pořízenými v 7 různých rozlišeních v závislosti na počtu tváří v těchto obrazech. Poté bylo ze zmíněné sady testů vytvořeno 7 křivek, jež analogicky znázorňují rychlosti dosažené během procesu souběžného sledování a detekování tváří probíhajícího nad obrazy pořízenými v 7 různých rozlišeních v závislosti na počtu tváří v těchto obrazech. Tato sedmice křivek je graficky zobrazena v souhrnném grafu na obrázku P-13 v příloze 2. Z těchto dvou sedmic opět vzešly dvě průměrové křivky. První znázorňuje rychlost OpenCV detekce tváří v obraze s průměrným rozlišením v závislosti na počtu tváří v těchto obrazech. Druhá znázorňuje rychlost procesu souběžného sledování a detekování tváří probíhajícího nad obrazy s průměrným rozlišením v závislosti na počtu tváří v těchto obrazech. Obě křivky jsou graficky znázorněny na obrázku 7.12.



Obrázek 7.12: Graf závislosti rychlosti souběžného sledování na počtu sledovaných tváří v obraze.

Ze stejné sady testů bylo vytvořeno ještě dalších 7 křivek, viz obrázek P-14 v příloze 2, jež znázorňují urychlení procesu souběžného sledování a detekování tváří oproti OpenCV detekci tváří nad obrazy se 7 různými rozlišeními v závislosti na počtu tváří. Těchto 7 křivek vzniklo analogickým postupem jako šestice z obrázku P-3 v příloze 2. Průměrem zmíněných sedmi křivek nakonec vznikla ještě poslední křivka, jež znázorňuje urychlení procesu souběžného sledování a detekování tváří v obrazech s průměrným rozlišením v závislosti na počtu tváří v těchto obrazech. Zmíněná průměrová křivka je znázorněna na následujícím obrázku 7.13.

Obrázek 7.13: Graf závislosti přínosu souběžného sledování na počtu sledovaných tváří v obraze.

7.3 Shrnutí provedeného testování

Z výsledků testů prezentovaných podkapitolou 7.2.1 lze konstatovat, že rychlosti obou testovaných detektorů v rámci podkapitoly 7.2.1 byly více závislé na rozlišení vstupního obrazu, než na počtu tváří ve vstupním obraze. Přes nízkou závislost na počtu tváří ve vstupním obraze, však lze usoudit, že čím více tváří se vyskytovalo ve vstupním obraze, tím většího urychlení dosahovala optimalizovaná detekce oproti OpenCV detekci. Naproti tomu přínos optimalizovaného detektoru se s konstantně rostoucím rozlišením nezvyšoval konstantně. Přínos optimalizovaného detektoru namísto konstantního růstu dosáhl svého maxima při rozlišeních 800x600 a 1024x768 pixelů a od rozlišení 1280x960 se již téměř neměnil.

Z grafických výsledků uvedených v podkapitole 7.2.1.2 a 7.2.1.3 lze usoudit, že rychlost optimalizovaného detektoru byla vysoce závislá na volbě kroku procházení podoken. Dále z těchto dvou podkapitol můžeme soudit, že nárůst zrychlení optimalizovaného detektoru v závislosti na velikosti kroku procházení podoken s rostoucím krokem měl přibližně stejný průběh pro všechna rozlišení vstupního obrazu. Naproti tomu, přínos optimalizovaného detektoru oproti OpenCV detektoru v závislosti na velikosti kroku nabýval na významu s vyšším počtem tváří ve vstupním obraze, konkrétně v obraze se 170 tvářemi měla křivka zrychlení optimalizovaného detektoru oproti OpenCV detekci nejstrmější průběh. Z těchto dvou podkapitol je také patrné, že s rostoucím krokem klesala úspěšnost optimalizovaného detektoru.

Z podkapitoly 7.2.2 lze soudit, že proces souběžného sledování a detekování tváří je daleko více závislý na počtu tváří ve vstupním obraze, než na rozlišení vstupního obrazu. Konkrétně lze soudit, že přínos procesu souběžného sledování a detekování tváří s rostoucím rozlišením neustále rostl, zatímco s přibývajícím počtem tváří ve vstupním obraze tento přínos naopak klesal.

S ohledem na všechny provedené testy je tak možné do tohoto odstavce napsat, že proces kombinující sledování tváří s optimalizovaným detektorem přináší do navržené knihovny nejvyšší urychlení ze všech navržených řešení. Zmíněným procesem souběžného sledování a detekování tváří bylo dosaženo např. rychlosti cca **53 fps** pro vstupní obraz v průměrném rozlišení s jednou tváří, nebo např. rychlosti cca **30 fps** pro vstupní obraz v průměrném rozlišení se dvěma tvářemi. Dále je vhodné zmínit, že i přes snižující se rychlost souběžného sledování a detekování tváří nebyl problém tímto řešením dosáhnout rychlosti cca **16 fps** pro obraz v průměrném rozlišení a 10 tvářemi. Pro srovnání je vhodné uvést, že OpenCV detekce dosáhla shodně 1.6 fps pro všechny tři zmíněné vstupní obrazy, tedy pro obrazy v průměrném rozlišení s jednou, se dvěma a s 10 tvářemi. Z celého testování je dále patrné, že samostatně běžící optimalizovanou detekcí bylo možné detekovat tváře ve vstupním obraze s průměrným počtem tváří a s průměrným rozlišením s rychlostí cca 2 fps, naproti tomu OpenCV detekce tváří ve stejném obraze dosáhla rychlosti 1.5 fps, což znamená, že optimalizovaná detekce z kapitoly 5.1.6 vyhodnotila průměrný snímek o cca **33 % rychleji** v porovnání s OpenCV detekcí.

8 Závěr

Prvním cílem diplomové práce bylo nastudování algoritmů používaných při detekcích obličejů ve videu. Toto studium proběhlo ve velice širokém rozsahu, což vedlo k prozkoumání nejen algoritmů pro detekci obličejů ve videu, ale také algoritmů pro sledování tváří ve videu a algoritmů pro detekci lidské kůže ve videu. Druhým cílem diplomové práce bylo popsání vybraných nastudovaných algoritmů. Tento cíl byl úspěšně splněn v rámci kapitol dvě, tři a čtyři. Třetím cílem diplomové práce pak bylo využití znalostí nabytých v rámci prvních dvou cílů diplomové práce k vytvoření návrhu knihovny, jež by bylo možné použít k detekci obličejů ve videu. Návrh takové knihovny byl popsán v rámci kapitoly 5.

Čtvrtým cílem bylo pro navrženou knihovnu nalézt možné optimalizace. Nejprve byla snaha tento cíl splnit optimalizací pomocí SIMD instrukcí. Optimalizace pomocí SIMD instrukcí sice přinesly cca 35% zrychlení detektoru v porovnání s OpenCV detektorem, nicméně na real-time běh samotného detektoru to nestačilo. Proto přišla na řadu další snaha, spojení optimalizovaného detektoru se sledováním tváří, což se ukázalo jako velice dobrá cesta k dosažení real-time detekování/sledování tváří ve videu pro naprostou většinu vstupních video sekvencí v libovolném rozlišení. Systémem souběžného sledování/detekování bylo bez problémů dosaženo např. výborné rychlosti cca **45 fps** pro jednu tvář ve vstupní videosekvenci s rozlišením 1280x960 pixelů. Jediná nevýhoda kombinace optimalizovaného detektoru se sledováním je závislost tohoto řešení na počtu tváří ve scéně. Zmíněná nevýhoda však není vyloženě kritická, jelikož např. i pro obrazy s průměrným rozlišením a 10 tvářemi nebyl problém dosáhnout 16 fps, což může být stále považováno za real-time běh. Proto může být i čtvrtý cíl diplomové práce považován za úspěšně dosažený. Posledním cílem diplomové práce byla praktická implementace navržené knihovny, což se opět podařilo a důkazem jsou kapitoly 6 a 7.

Na základě této kapitoly je tak možné konstatovat, že všechny cíle diplomové práce se podařilo úspěšně splnit a jako výsledek celé práce tak vznikla rychlá knihovna pro detekci obličejů ve videu klasifikující na základě velice robustního OpenCV klasifikátoru, jež dosahuje perfektní úspěšnosti při klasifikacích. Pokud by měla být tato knihovna v budoucnu někdy rozšiřována, bylo by možné zaměřit se např. na identifikaci osob dle nalezených obličejů.

Za hlavní přínos této diplomové práce může být považována samotná knihovna, jež byla realizovaná v rámci praktické části této diplomové práce. Za další přínosy diplomové práce pak můžou být považovány demonstrace nestandardního použití SIMD instrukcí a také demonstrace použití Kalmanova filtru pro účely sledování obličejů.

Literatura

[1] ŽÁRA, J., B. BENEŠ, J. SOCHOR a P. FELKEL. *Moderní počítačová grafika*. 2. přeprac. vyd. Brno: Computer Press, 2004. ISBN 80-251-0454-0.

[2] CMYK. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 17. 1. 2006, last modified on 6. 10. 2011 [cit. 2012-01-06]. Dostupné z: http://cs.wikipedia.org/wiki/CMYK

[3] YUV. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 29. 9. 2006, last modified on 27. 3. 2011 [cit. 2012-01-06]. Dostupné z: http://cs.wikipedia.org/wiki/YUV

[4] FULTON, Wayne. *Indexed Color Palettes* [online]. ©1997-2010 [cit. 2012-01-06]. Dostupné z: http://www.scantips.com/palettes.html

[5] VEZHNEVETS, V., V. SAZONOV and A. ANDREEVA. A survey on pixel-based skin color detection techniques. *Cybernetics* [online]. 2003, vol. 85 [2012-01-06], pages 85-92. Dostupný z: http://www.mendeley.com/research/a-survey-on-pixelbased-skin-color-detection-techniques/

[6] GÓMEZ, G. and E. Morales. Automatic feature construction and a simple rule induction algorithm for skin detection. *The Nineteenth ICML workshop on Machine Learning in Computer Vision* [online]. July 9, 2002 [cit. 2012-01-06], Sydney, Australia, pages 31-38. Dostupné z: http://www.cse.unsw.edu.au/~icml2002/workshops/MLCV02/MLCV02-Morales.pdf

[7] VLACH, J., J. PŘINOSIL. Lokalizace obličeje v obraze s komplexním pozadím. *Elektrorevue* [online]. 11. 4. 2007 [cit. 2012-01-06]. ISSN 1213-1539. Dostupné z: http://www.elektrorevue.cz/file.php?id=200000112-66258671fa

[8] YUV. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 29. 9. 2002, last modified on 3. 1. 2012 [cit. 2012-01-06]. Dostupné z: http://en.wikipedia.org/wiki/YUV

[9] BASILIO, J. A. M. et al. Explicit Image Detection using YCbCr Space Color Model as Skin Detection. *The 5th WSEAS International Conference on Computer Engineering and Applications (CEA '11)* [online]. January 29-31, 2011 [cit. 2012-01-06], Puerto Morelos, Mexico, pages. 123-128. ISSN: 1792-7269/ 1792-7722, ISBN: 978-960-474-270-7. Dostupné z: http://www.wseas.us/e-library/conferences/2011/Mexico/CEMATH/CEMATH-29.pdf

[10] MAHMOUD, T. M. A New Fast Skin Color Detection Technique. *World Academy of Science, Engineering and Technology* [online]. 2008, vol2012-01-06]. Dostupný z: http://www.waset.org/journals/waset/v43/v43-91.pdf

[11] VIOLA, P. and M. JONES. Robust Real-time Object Detection. *Second international workshop on statistical and computational theories of vision – modeling, learning, computing, and sampling* [online]. July 13, 2001 [cit. 2012-01-06], Vancouver, Canada. Dostupné z: http://research.microsoft.com/en-us/um/people/viola/Pubs/Detect/violaJones_IJCV.pdf

[12] PŘINOSIL, Jiří. Využití detektoru Viola-Jones pro lokalizaci obličeje a očí v barevných obrazech. *Elektrorevue* [online]. 21. 8. 2008 [cit. 2012-01-06]. ISSN 1213-1539. Dostupné z: http://www.elektrorevue.cz/file.php?id=200000238-97bc998b67

[13] HRADIŠ, Michal. *Klasifikace a rozpoznávání: Boosting* [online prezentace]. 6. 4. 2009 [cit. 2012-01-06], Brno: Ústav počítačové grafiky a multimédií, FIT, VUT. Dostupný z: http://www.fit.vutbr.cz/study/courses/IKR/public/prednasky/07_adaboost/IKR-Boosting.pdf

[14] MOLNÁR, Karol. Úvod do problematiky umělých neuronových sítí. *Elektrorevue* [online]. 22. 2. 2008 [cit. 2012-01-06]. ISSN 1213-1539. Dostupné z: http://www.elektrorevue.cz/clanky/00013/index.html

[15] SCHWRAZ, Petr. *Klasifikace a rozpoznávání: Umělé neuronové sítě a Support Vector Machines* [online prezentace]. 19. 5. 2009 [cit. 2012-01-06], Brno: Ústav počítačové grafiky a multimédií, FIT, VUT. Dostupný z: www.fit.vutbr.cz/study/courses/IKR/public/prednasky/06_nn_svm/nn_svm3.ppt

[16] Artificial neuron. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 25. 10. 2003, last modified on 4. 1. 2011 [cit. 2012-01-06]. Dostupné z: http://en.wikipedia.org/wiki/Artificial_neuron

[17] NOVÁK, M. et al. *Umělé neuronové sítě : teorie a aplikace*. 1. vyd. Praha: C.H. Beck, 1998. ISBN 80-7179-132-6.

[18] ŽIŽKA, Jindřích. *Support vector machines (SVM)* [online prezentace]. 21. 10. 2005 [cit. 2012-01-06], Brno: Katedra teorie programování, Fakulta Informatiky, MU. Dostupný z: http://is.muni.cz/el/1433/podzim2006/PA034/09_SVM.pdf

[19] Perceptron. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 22. 1. 2003, last modified on 12. 12. 2011 [cit. 2012-01-06]. Dostupné z: http://en.wikipedia.org/wiki/Perceptron

[20] *LBPShort − Department of Computer Science and Engineering* [online]. ©2009, last modified on 18.11.2011 [cit. 2011-12-23]. Dostupné z: http://www.cse.oulu.fi/CMV/Research/LBP/LBPShort

[21] PIETIKÄINEN, Matti. Local Binary Patterns. *Scholarpedia, the peer-reviewed open-access encyclopedia* [online]. 14. 7. 2009, last modified on 21. 10. 2011 [cit. 2012-01-06]. ISSN 1941-6016. Dostupné z: http://www.scholarpedia.org/article/Local_Binary_Patterns

[22] LÁNÍK, A. a J. ZUZAŇÁK. *Klasifikace a rozpoznávání: Extrakce obrazových příznaků* [online prezentace]. 30. 3. 2011 [cit. 2012-01-06], Brno: Ústav počítačové grafiky a multimédií, FIT, VUT. Dostupný z: http://www.fit.vutbr.cz/study/courses/IKR/public/prednasky/04_obrazove_priznaky/ikr-obrazove-priznaky-2011.pdf

[23] AHONEN, T., A. HADID and M. PIETIKÄINEN. Face Description with Local Binary Patterns: Application to Face Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. December 2006, vol. 28, issue 12, pages 2037 - 2041 [cit. 2012-01-06]. ISSN 0162-8828. Dostupné z: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1717463

[24] HEIKKILÄ, M. and M. PIETIKÄINEN. A texture-based method for modeling the background and detecting moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. April 2006, vol. 28, issue 4, pages 657 - 662 [cit. 2012-01-06]. ISSN 0162-8828. Dostupné z: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1597122

[25] Statistical classification. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 7. 3. 2005, last modified on 16. 12. 2011 [cit. 2012-01-06]. Dostupné z: http://en.wikipedia.org/wiki/Statistical_classification

[26] BURGET, Lukáš. *Klasifikace a rozpoznávání: Lineární klasifikátory* [online prezentace]. 30. 5. 2011 [cit. 2012-01-06], Brno: Ústav počítačové grafiky a multimédií, FIT. Dostupný z: http://www.fit.vutbr.cz/study/courses/IKR/public/prednasky/05_lin_klasifikatory/IKR5.ppt

[27] Linear classifier. In: *Wikipedia: the free encyclope*dia [online]. St. Petersburg (Florida): Wikipedia Foundation, 3. 10. 2002, last modified on 12. 08. 2011 [cit. 2012-01-06]. Dostupné z: http://en.wikipedia.org/wiki/Linear_classifier

[28] Machine learning. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 25. 5. 2003, last modified on 28. 12. 2011 [cit. 2012-01-06]. Dostupné z: http://en.wikipedia.org/wiki/Machine_learning

[29] Sony Cyber-shot DSCW180 Digital Camera - Black. *Amazon.com* [online]. Seattle (Washington): Amazon S.à.r.l, 1. 7. 2006, [cit. 2012-01-06]. Dostupné z: http://www.amazon.co.uk/Sony-Cyber-shot-DSCW180-Digital-Camera/dp/B002A9JDKM

[30] YCbCr. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 12. 4. 2004, last modified on 25. 12. 2011 [cit. 2012-01-06]. Dostupné z: http://en.wikipedia.org/wiki/YCbCr

[31] ORSÁG, Filip. *Pokročilé asemblery PAS: Studijní opora* [online]. 31. 10. 2006 [cit. 2012-01-06], Brno: Ústav inteligentních systémů, FIT, VUT. Dostupné z: https://www.fit.vutbr.cz/study/courses/IPA/private/PAS.pdf

[32] KUNTZ, Noah. *OpenCV Tutorial 1* [online]. 17. 12. 2008, last modified on 22. 12. 2009 [2012-01-06]. Dostupné z: http://dasl.mem.drexel.edu/~noahKuntz/openCVTut1.html

[33] WELCH, Greg and Gary Bishop. *An Introduction to the Kalman Filter* [online]. 12. 8. 2001, last modified on 24. 6. 2006 [cit. 2012-04-29]. Dostupné z: http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf

[34] *Object tracking using Kalman filter (Matlab)* [online blog]. 3. 5. 2011 [cit. 2012-05-09]. Dostupné z: http://blog.cordiner.net/2011/05/03/object-tracking-using-a-kalman-filter-matlab/

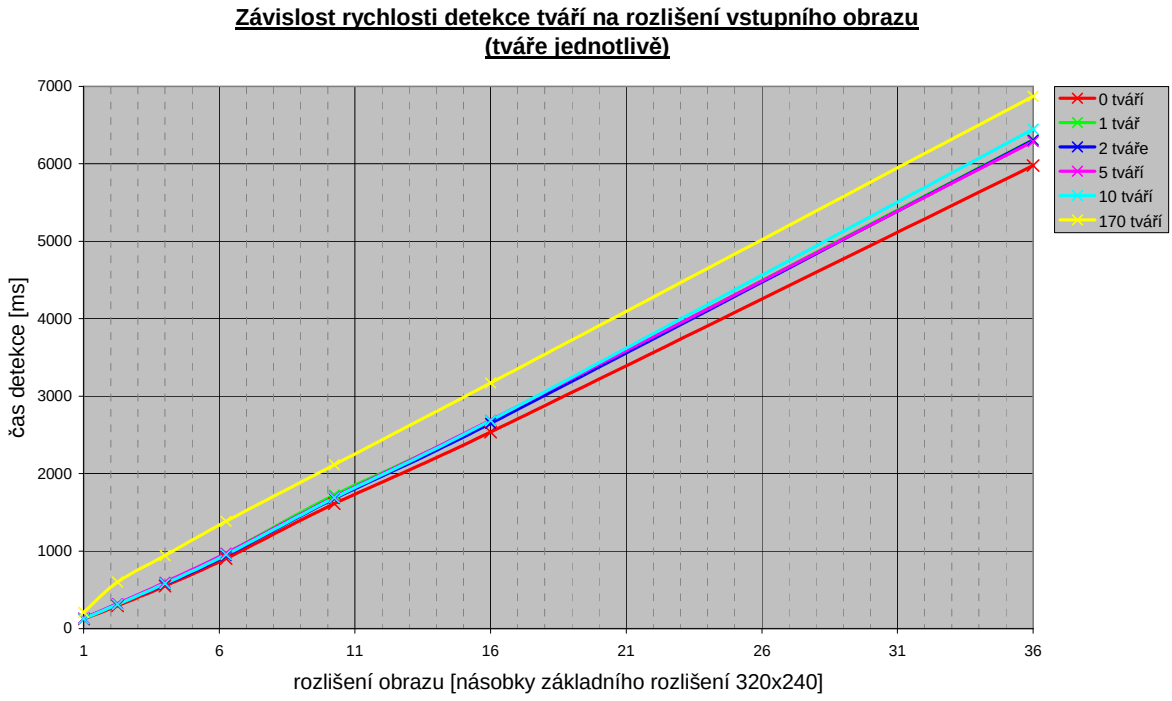
[35] WINKLER, Zbyněk. *Měření rychlosti* [online]. 10. 11. 2005 [cit. 2012-05-09] . Dostupné z http://robotika.cz/guide/filtering/en

Příloha 1

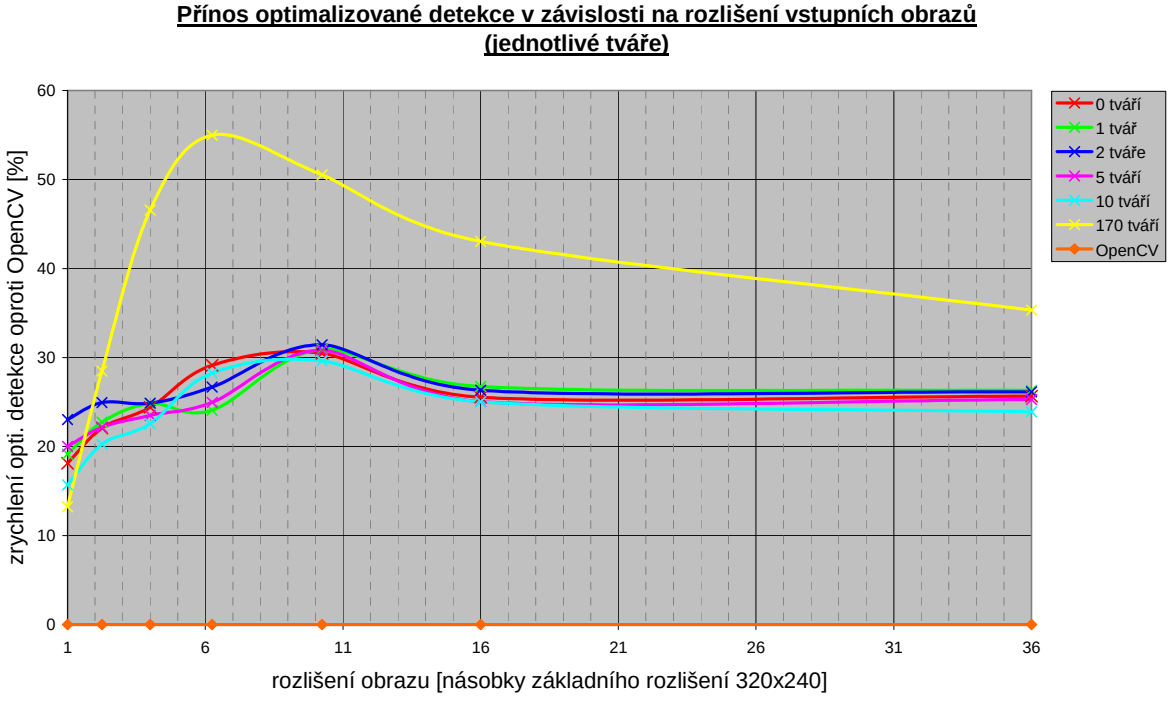
```
<opencv_storage>
<haarcascade_frontalface_tree_alt type_id="opencv-haar-classifier">
  <size>20 20</size>
  <stages>
    <_>
      <!-- stage 0 -->
      <trees>
        <_>
          <!-- tree 0 -->
          <_>
            <!-- root node -->
            <feature>
              <rects>
                <_>2 7 14 4 -1.</_>
                <_>2 9 14 2 2.</_></rects>
              <tilted>0</tilted></feature>
            <threshold>3.7895569112151861e-003</threshold>
            <left_val>-0.9294580221176148</left_val>
            <right_val>0.6411985158920288</right_val></_></_>
        <_>
          <!-- tree 1 -->
          <_>
            <!-- root node -->
            <feature>
              <rects>
                <_>1 2 18 4 -1.</_>
                <_>7 2 6 4 3.</_></rects>
              <tilted>0</tilted></feature>
            <threshold>0.0120981102809310</threshold>
            <left_val>-0.7181009054183960</left_val>
            <right_val>0.4714100956916809</right_val></_></_>
        <_>
          <!-- tree 2 -->
          <_>
            <!-- root node -->
            <feature>
              <rects>
                <_>5 5 9 5 -1.</_>
                <_>8 5 3 5 3.</_></rects>
              <tilted>0</tilted></feature>
            <threshold>1.2138449819758534e-003</threshold>
            <left_val>-0.7283161282539368</left_val>
            <right_val>0.3033069074153900</right_val></_></_></trees>
      <stage_threshold>-1.3442519903182983</stage_threshold>
      <parent>-1</parent>
      <next>-1</next>
    </_>

      .
      .
      .

  </stages>
</haarcascade_frontalface_alt>
</opencv_storage>
```

Obrázek P-1: Ukázka jednoho stupně kaskády OpenCV klasifikátoru, konkrétně klasifikátoru *haarcascade_frontalface_alt.xml*.

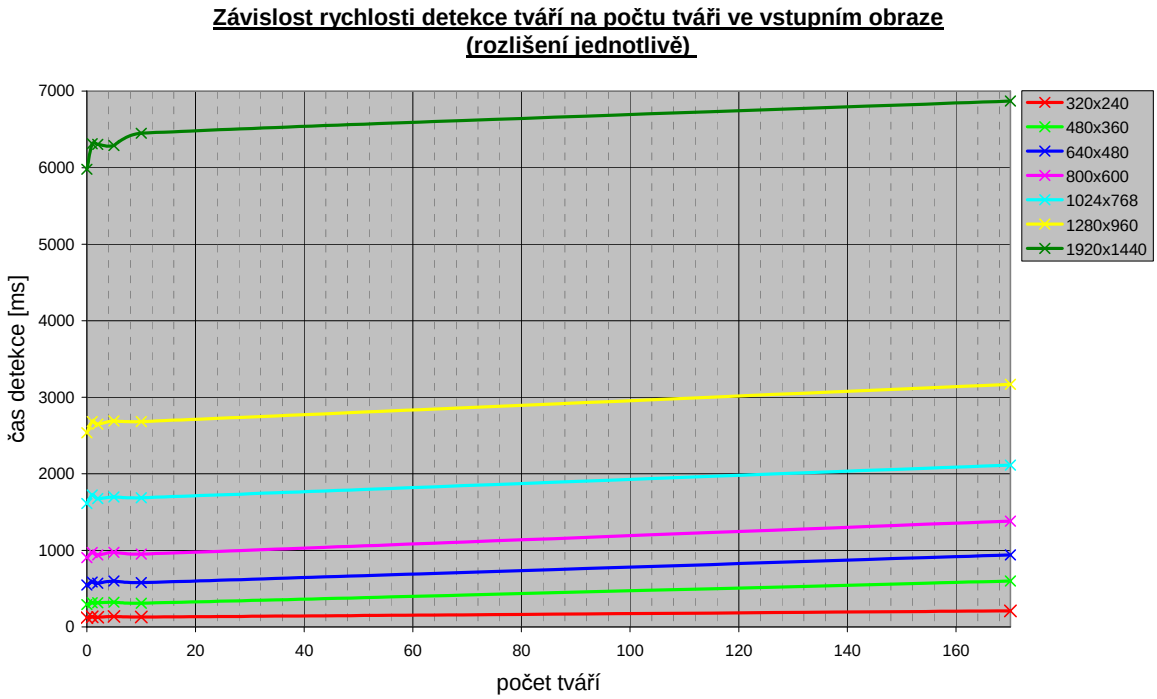
Příloha 2



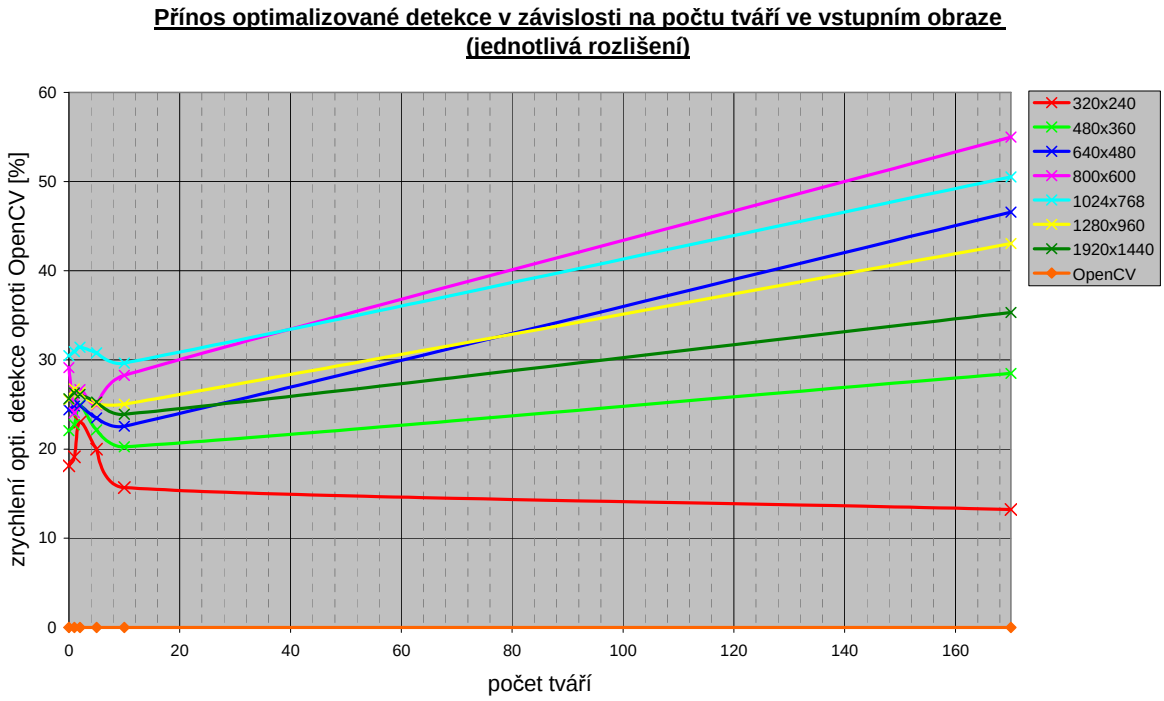
Obrázek P-2: Graf závislosti rychlosti detekce tváří na rozlišení vstupního obrazu pro jednotlivé počty tváří odděleně.



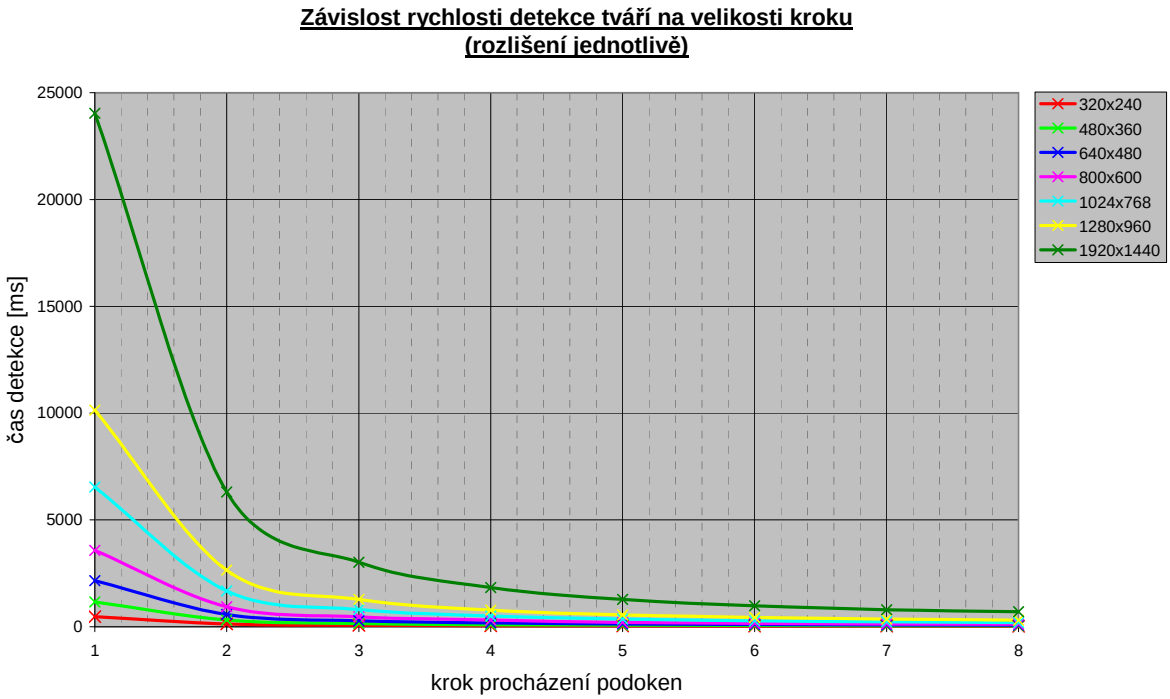
Obrázek P-3: Graf závislosti přínosu optimalizované detekce tváří na rozlišení vstupního obrazu. pro jednotlivé počty tváří odděleně.



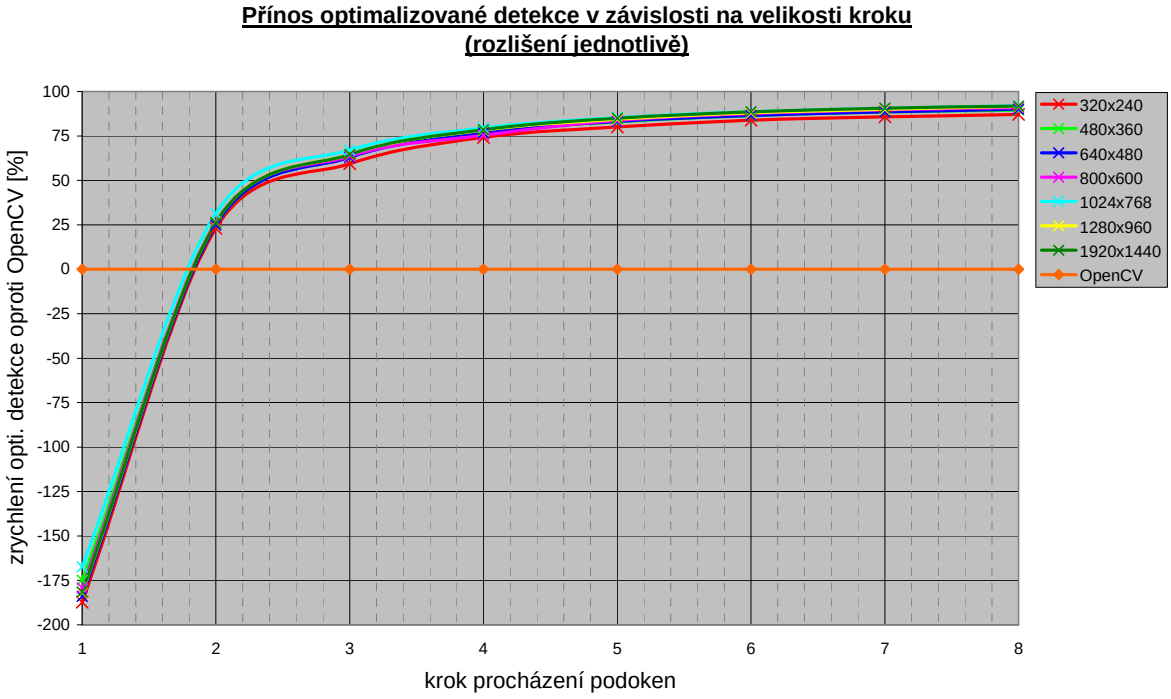
Obrázek P-4: Grafické znázornění závislosti rychlosti detekce tváří na počtu tváří pro jednotlivá rozlišení odděleně



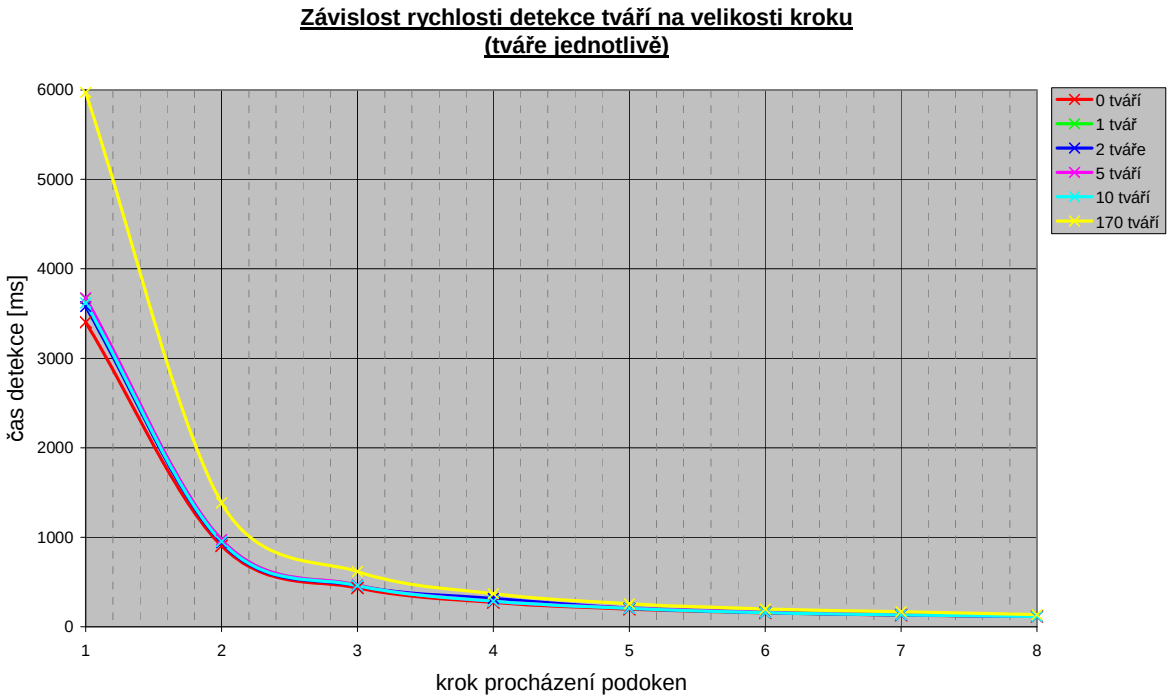
Obrázek P-5: Graf závislosti přínosu optimalizované detekce tváří na počtu tváří ve vstupním obraze. pro jednotlivá rozlišení odděleně.



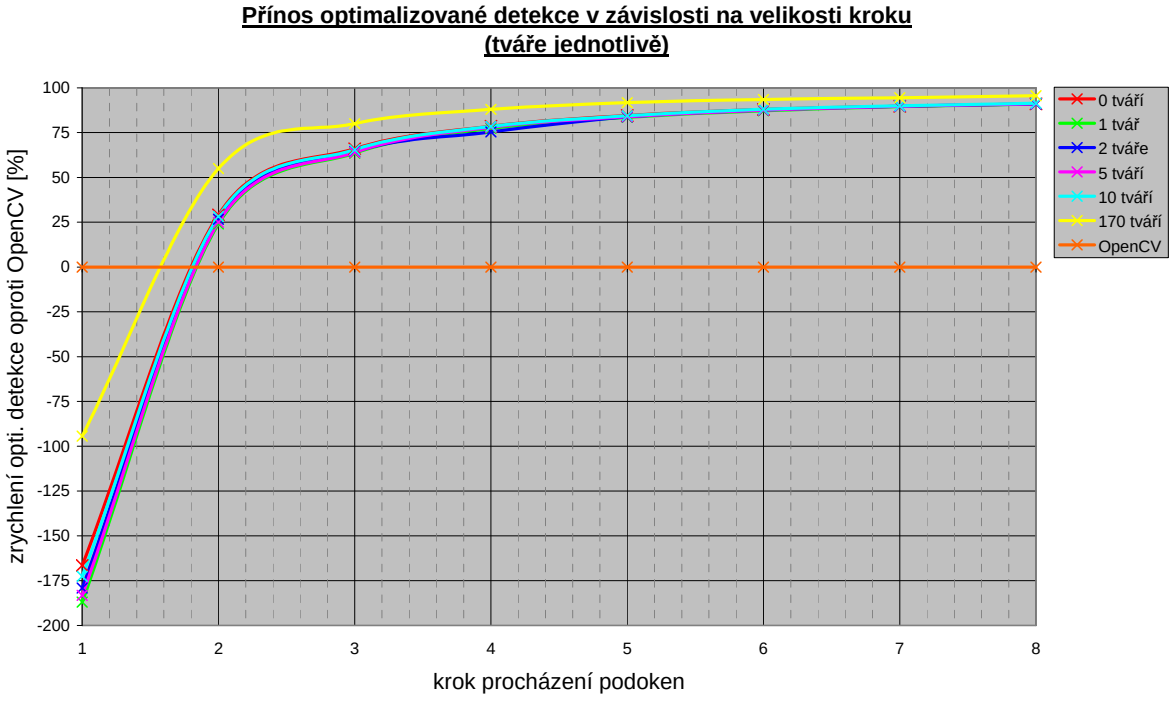
Obrázek P-6: Graf závislosti rychlosti detekce tváří v obraze se dvěma tvářemi na kroku. pro jednotlivá rozlišení odděleně.



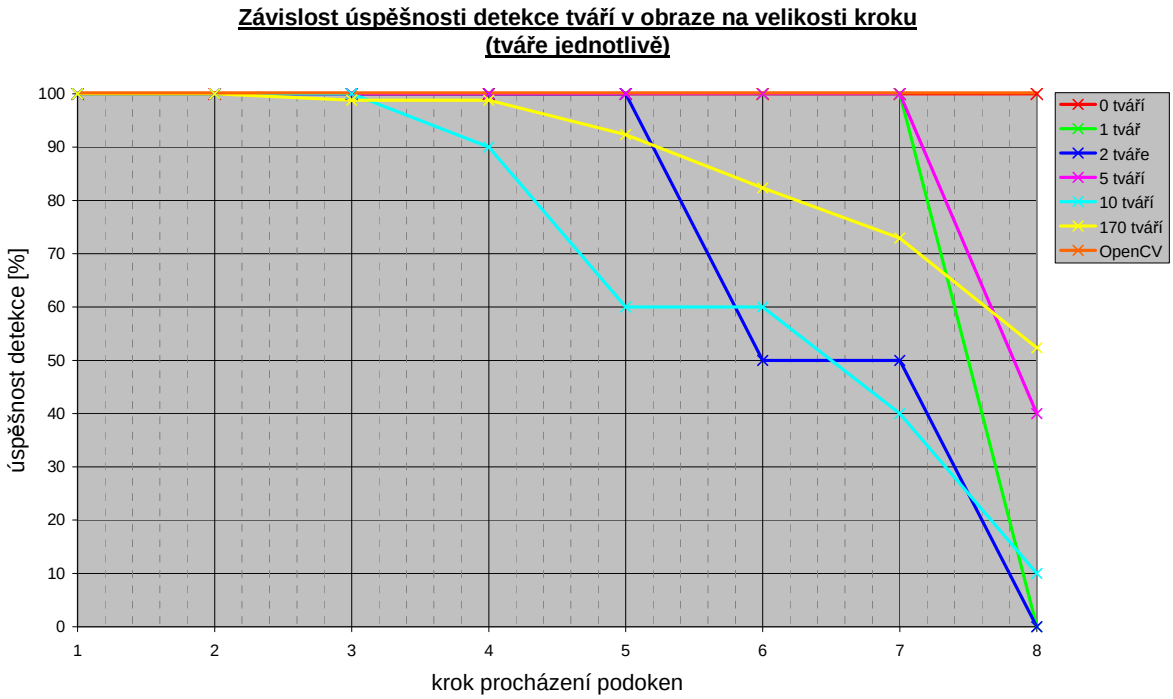
Obrázek P-7: Graf závislosti přínosu optimalizované detekce tváří v obraze se dvěma tvářemi pro jednotlivá rozlišení odděleně.



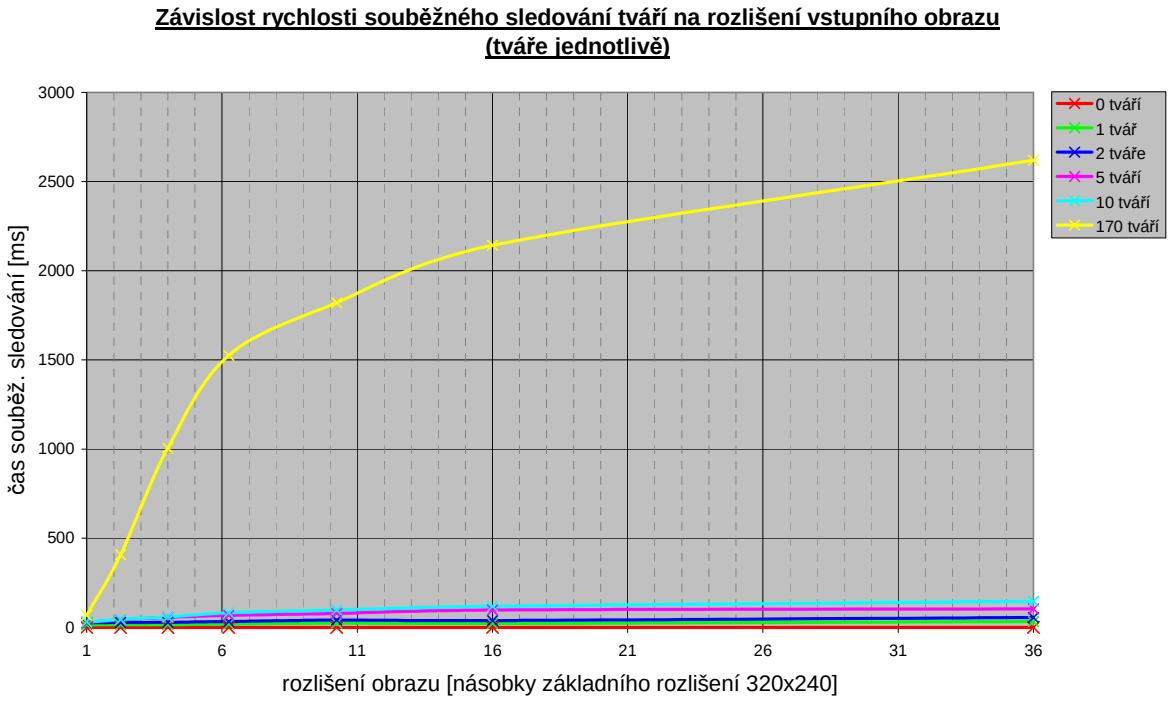
Obrázek P-8: Graf závislosti rychlosti detekce tváří v obraze s rozlišením 800x600 pixelů na kroku pro jednotlivé počty tváří odděleně.



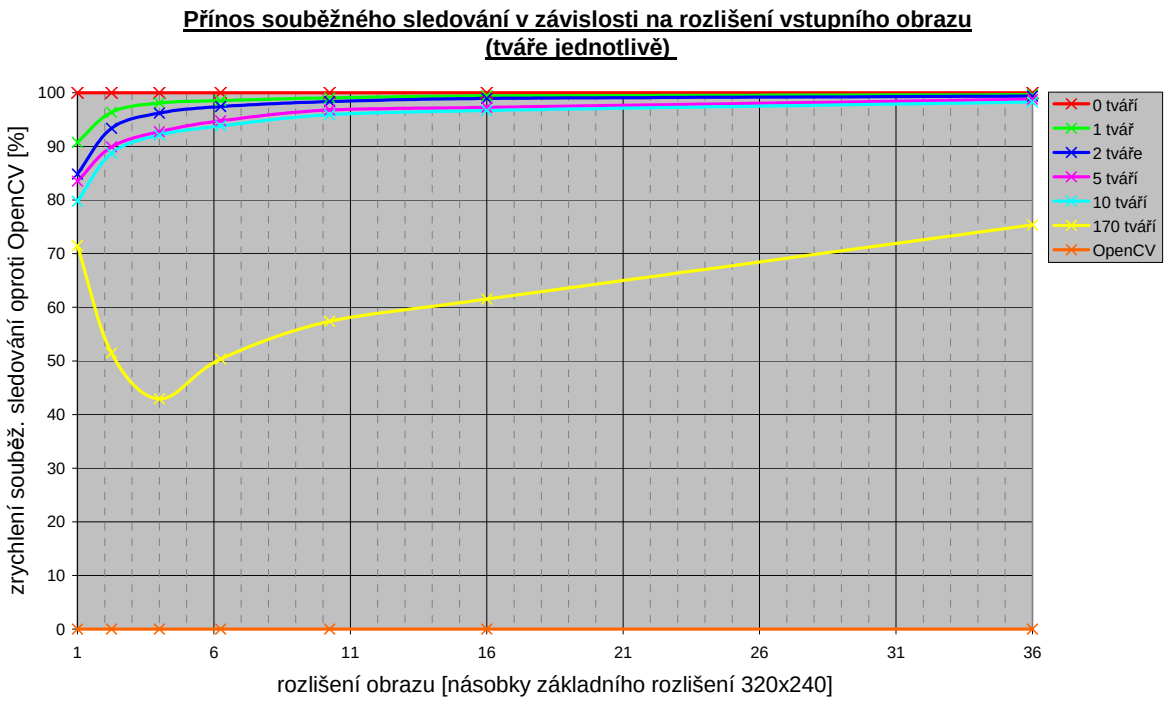
Obrázek P-9: Graf závislosti přínosu optimalizované detekce tváří v obraze s rozlišením 800x600 pixelů na kroku pro jednotlivé počty tváří odděleně.



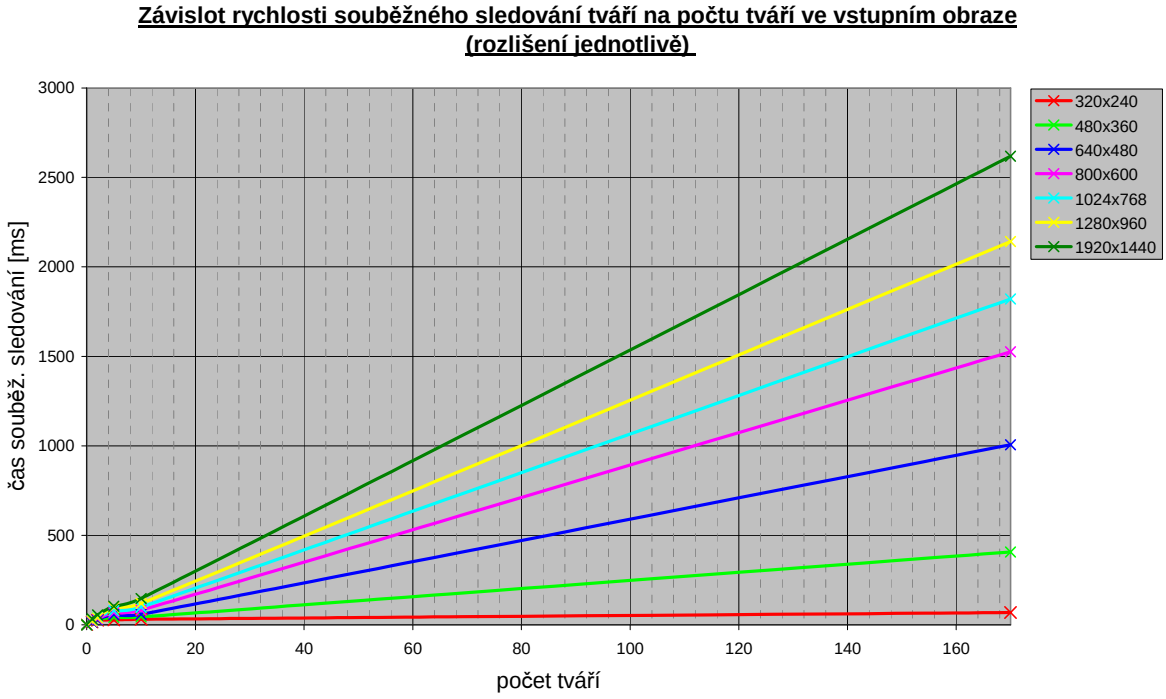
Obrázek P-10: Graf závislosti úspěšnosti detekce tváří v obraze s rozlišením 800x600 pixelů na kroku pro jednotlivé počty tváří odděleně.



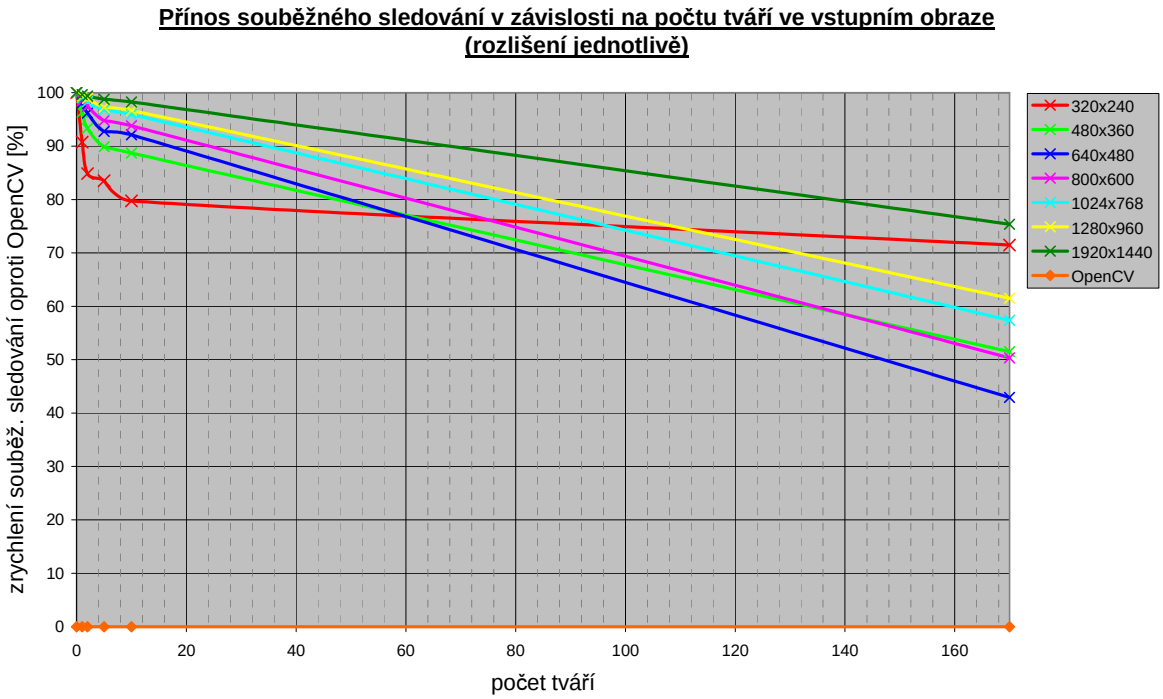
Obrázek P-11: Graf závislosti rychlosti souběžného sledování na rozlišení vstupního obrazu pro jednotlivé počty tváří odděleně.



Obrázek P-12: Graf závislosti přínosu souběžného sledování na rozlišení vstupního obrazu pro jednotlivé počty tváří odděleně.



Obrázek P-13: Graf závislosti rychlosti souběžného sledování na počtu sledovaných tváří v obraze pro jednotlivá rozlišení odděleně.



Obrázek P-14: Graf závislosti přínosu souběžného sledování na počtu sledovaných tváří v obraze pro jednotlivá rozlišení odděleně.

Příloha 3

			Počet tváří v obraze					
		Rozlišení	0	1	2	5	10	170
Čas zpracování jednoho snímku [ms]	OpenCV detektor s klasifikátorem *alt*	320x240	149	162	165	170	153	242
		480x360	376	409	421	415	390	839
		640x480	725	781	760	783	749	1763
		800x600	1277	1279	1282	1298	1327	3074
		1024x768	2320	2496	2447	2454	2399	4275
		1280x960	3406	3661	3597	3589	3580	5566
		1920x1440	8042	8563	8536	8418	8475	10626
	Standardní detektor s opti. průchodem podoken (5.1.4)	320x240	164	173	172	179	164	272
		480x360	392	418	413	429	407	771
		640x480	732	788	771	804	762	1188
		800x600	1197	1285	1256	1280	1241	1743
		1024x768	2120	2257	2234	2251	2201	2689
		1280x960	3339	3558	3503	3547	3500	4051
		1920x1440	7774	8284	8278	8277	8294	8967
	Optimalizovaný detektor (5.1.6)	320x240	122	131	127	136	129	210
		480x360	293	316	316	323	311	600
		640x480	548	587	571	599	580	942
		800x600	905	971	940	974	952	1384
		1024x768	1613	1724	1678	1698	1688	2115
		1280x960	2536	2682	2650	2690	2684	3170
		1920x1440	5979	6312	6304	6290	6449	6872
	Opti. detektor při souběžném sledování na základě *x* a *v*	320x240	176	187	187	187	187	249
		480x360	359	421	422	448	437	837
		640x480	767	802	795	780	728	2073
		800x600	1044	1175	1169	1231	1159	3276
		1024x768	1792	1792	2024	2093	2005	3752
		1280x960	2877	3685	3503	3348	3484	4353
		1920x1440	7390	8976	8793	8245	7345	8219
	Sledování na základě *x*, a *v* při souběžném opti. detektoru	320x240	0	15	25	28	31	69
		480x360	0	15	28	42	44	407
		640x480	0	15	29	57	59	1006
		800x600	0	19	33	68	82	1526
		1024x768	0	24	41	79	98	1821
		1280x960	0	22	39	98	119	2142
		1920x1440	0	33	55	104	147	2619

Tabulka P-1: Výsledky první a čtvrté sady testů (vstupní obrazy pořízené v 7 různých rozlišeních obsahující 6 různých počtů tváří).

			Rozlišení obrazu						
		krok	320x240	480x360	640x480	800x600	1024x768	1280x960	1920x1440
Čas detekce tváří v jednom snímku [ms]	OpenCV detektor s klasifikátorem *alt*	1	165	421	760	1282	2447	3597	8536
		2	165	421	760	1282	2447	3597	8536
		3	165	421	760	1282	2447	3597	8536
		4	165	421	760	1282	2447	3597	8536
		5	165	421	760	1282	2447	3597	8536
		6	165	421	760	1282	2447	3597	8536
		7	165	421	760	1282	2447	3597	8536
		8	165	421	760	1282	2447	3597	8536
	Standardní detektor s opti. průchodem podoken (5.1.4)	1	643	1574	2954	4858	8946	13703	32223
		2	172	413	771	1256	2234	3503	8278
		3	87	201	371	607	1049	1673	3903
		4	55	123	227	378	632	1000	2334
		5	40	89	161	258	446	699	1595
		6	32	70	124	195	340	531	1207
		7	27	58	103	160	274	429	976
		8	24	51	88	135	230	357	842
	Optimalizovaný detektor (5.1.6)	1	474	1157	2158	3579	6543	10144	24033
		2	127	316	571	940	1678	2650	6304
		3	67	153	281	467	807	1274	3019
		4	43	96	177	316	500	780	1830
		5	33	72	128	208	359	560	1280
		6	27	57	102	159	277	432	981
		7	24	48	87	132	231	360	801
		8	21	44	75	116	196	302	705

Tabulka P-2: Výsledky druhé sady testů (vstupní obrazy se 2 tvářemi v 7 různých rozlišení pro 8 různých kroků procházení podoken).

			Počet tváří v obraze					
		krok	0	1	2	5	10	170
Čas detekce tváří v jednom snímku [ms]	OpenCV detektor s klasifikátorem *alt*	1	1277	1279	1282	1298	1327	3074
		2	1277	1279	1282	1298	1327	3074
		3	1277	1279	1282	1298	1327	3074
		4	1277	1279	1282	1298	1327	3074
		5	1277	1279	1282	1298	1327	3074
		6	1277	1279	1282	1298	1327	3074
		7	1277	1279	1282	1298	1327	3074
		8	1277	1279	1282	1298	1327	3074
	Standardní detektor s opti. průchodem podoken (5.1.4)	1	4603	4964	4858	4935	4782	7470
		2	1197	1285	1256	1280	1241	1743
		3	568	613	597	609	589	789
		4	349	372	378	371	360	475
		5	246	261	258	259	252	318
		6	186	198	195	197	193	240
		7	158	162	160	160	158	194
		8	131	136	135	136	135	159
	Optimalizovaný detektor (5.1.6)	1	3404	3670	3579	3676	3618	5973
		2	905	971	940	974	952	1384
		3	434	466	457	467	459	616
		4	273	289	316	285	286	370
		5	197	210	208	209	206	256
		6	154	164	159	162	158	199
		7	131	134	132	134	133	169
		8	114	116	116	116	115	136

Tabulka P-3: Výsledky třetí sady testů - rychlost detekce (vstupní obrazy v rozlišení 800x600 pixelů obsahující 6 různých počtů tváří pro 8 různých kroků procházení podoken).

			Počet tváří v obraze					
		krok	0	1	2	5	10	170
Úspěšnost detektoru [úspěšné / neúspěšné detekce]	OpenCV detektor s klasifikátorem *alt*	1	0 / 0	1 / 0	2 / 0	5 / 0	10 / 0	170 / 1
		2	0 / 0	1 / 0	2 / 0	5 / 0	10 / 0	170 / 1
		3	0 / 0	1 / 0	2 / 0	5 / 0	10 / 0	170 / 1
		4	0 / 0	1 / 0	2 / 0	5 / 0	10 / 0	170 / 1
		5	0 / 0	1 / 0	2 / 0	5 / 0	10 / 0	170 / 1
		6	0 / 0	1 / 0	2 / 0	5 / 0	10 / 0	170 / 1
		7	0 / 0	1 / 0	2 / 0	5 / 0	10 / 0	170 / 1
		8	0 / 0	1 / 0	2 / 0	5 / 0	10 / 0	170 / 1
	Standardní detektor s opti. průchodem podoken (5.1.4)	1	0 / 1	1 / 1	2 / 1	5 / 1	10 / 0	170 / 0
		2	0 / 0	1 / 0	2 / 0	5 / 0	10 / 0	170 / 1
		3	0 / 0	1 / 0	2 / 2	5 / 0	10 / 2	170 / 0
		4	0 / 1	1 / 1	2 / 0	5 / 1	9 / 1	168 / 2
		5	0 / 1	1 / 1	2 / 1	5 / 1	6 / 1	157 / 0
		6	0 / 0	1 / 1	1 / 0	5 / 0	6 / 1	140 / 0
		7	0 / 1	1 / 0	1 / 0	5 / 0	4 / 0	124 / 0
		8	0 / 0	0 / 1	0 / 0	2 / 2	1 / 0	89 / 0
	Optimalizovaný detektor (5.1.6)	1	0 / 1	1 / 1	2 / 1	5 / 1	10 / 0	170 / 0
		2	0 / 0	1 / 0	2 / 0	5 / 0	10 / 0	170 / 1
		3	0 / 0	1 / 0	2 / 2	5 / 0	10 / 2	170 / 0
		4	0 / 1	1 / 1	2 / 0	5 / 1	9 / 1	168 / 2
		5	0 / 1	1 / 1	2 / 1	5 / 1	6 / 1	157 / 0
		6	0 / 0	1 / 1	1 / 0	5 / 0	6 / 1	140 / 0
		7	0 / 1	1 / 0	1 / 0	5 / 0	4 / 0	124 / 0
		8	0 / 0	0 / 1	0 / 0	2 / 2	1 / 0	89 / 0

Tabulka P-4: Výsledky třetí sady testů - úspěšnost detekce (vstupní obrazy v rozlišení 800x600 pixelů obsahující 6 různých počtů tváří pro 8 různých kroků procházení podoken).